# firebase-rest-api

*Release 1.0.1*

**Asif Arman Rahman**

**Jul 30, 2022**

# CONTENTS

A simple python wrapper for Google's Firebase REST API's.

CHAPTER

ONE

# INSTALLATION

```
pip install firebase-rest-api
```

CHAPTER

TWO

# QUICK START

In order to use this library, you first need to go through the following steps:

1. Select or create a Firebase project from Firebase Console. *(guide)*
2. Register an Web App. *(guide)*

## 2.1 Example Usage

```
# Import Firebase REST API library
import firebase

# Firebase configuration
config = {
   "apiKey": "apiKey",
   "authDomain": "projectId.firebaseapp.com",
   "databaseURL": "https://databaseName.firebaseio.com",
   "projectId": "projectId",
   "storageBucket": "projectId.appspot.com",
   "messagingSenderId": "messagingSenderId",
   "appId": "appId"
}

# Instantiates a Firebase app
app = firebase.initialize_app(config)


# Firebase Authentication
auth = app.auth()

# Create new user and sign in
auth.create_user_with_email_and_password(email, password)
user = auth.sign_in_with_email_and_password(email, password)


# Firebase Realtime Database
db = app.database()

# Data to save in database
data = {
```

(continues on next page)

(continued from previous page)

```
    "name": "Robert Downey Jr.",
    "email": user.get('email')
}

# Store data to Firebase Database
db.child("users").push(data, user.get('idToken'))


# Firebase Storage
storage = app.storage()

# File to store in storage
file_path = 'static/img/example.png'

# Store file to Firebase Storage
storage.child(user.get('email')).child('uploaded-picture.png').put(file_path, user.get(
↪'idToken'))
```

CHAPTER

THREE

# DOCUMENTATION CONTENTS

## 3.1 Setup Project

Before you can add Firebase to your Python app, you need to create a Firebase project and register your app with that project. When you register your app with Firebase, you'll get a Firebase configuration object that you'll use to connect your app with your Firebase project resources.

### 3.1.1 Create a Firebase project

1. In the Firebase console, click **Add project**.
   - To add Firebase resources to an existing Google Cloud project, enter its project name or select it from the dropdown menu.
   - To create a new project, enter the desired project name. You can also optionally edit the project ID displayed below the project name.

   > **Attention:** Firebase generates a unique ID for your Firebase project based upon the name you give it. If you want to edit this project ID, you must do it now as it cannot be altered after Firebase provisions resources for your project. Visit Understand Firebase Projects to learn about how Firebase uses the project ID.

2. If prompted, review and accept the Firebase terms.
3. Click **Continue**.
4. (Optional) Set up Google Analytics for your project.

   **Note:** You can always set up Google Analytics later in the *integrations* tab of your Project settings.

5. Click **Create project** (or **Add Firebase**, if you're using an existing Google Cloud project).

Firebase automatically provisions resources for your Firebase project. When the process completes, you'll be taken to the overview page for your Firebase project in the Firebase console.

#### Setup Realtime Database

`databaseURL` key is not present by default in the Firebase configuration when an app is *registered*. It is recommended to setup database before *registering an app*.

### 3.1.2 Register your app

After you have a Firebase project, you can register your web app with that project.

1. In the center of the Firebase console's project overview page, click the **Web** icon () to launch the setup workflow.

   If you've already added an app to your Firebase project, click **Add app** to display the platform options.

2. Enter your app's nickname.
   This nickname is an internal, convenience identifier and is only visible to you in the Firebase console.

3. Click **Register app**.

4. Copy the Firebase configuration dict shown in the screen, and store it use to connect to your project later in code example part.
   The dict should be of the architecture shown below:

```
config = {
   "apiKey": "apiKey",
   "authDomain": "projectId.firebaseapp.com",
   "databaseURL": "https://databaseName.firebaseio.com",
   "projectId": "projectId",
   "storageBucket": "projectId.appspot.com",
   "messagingSenderId": "messagingSenderId",
   "appId": "appId"
}
```

5. Click **Continue to console**.

## 3.2 Integrate Firebase

You can integrate Firebase project into your Python app in two ways.

### 3.2.1 User based Authentication

For use with only user based authentication we can create the following configuration:

```
# Import Firebase REST API library
import firebase

# Firebase configuration
config = {
   "apiKey": "apiKey",
   "authDomain": "projectId.firebaseapp.com",
   "databaseURL": "https://databaseName.firebaseio.com",
   "projectId": "projectId",
```

(continues on next page)

(continued from previous page)

```
    "storageBucket": "projectId.appspot.com",
    "messagingSenderId": "messagingSenderId",
    "appId": "appId"
}

# Instantiates a Firebase app
firebaseApp = firebase.initialize_app(config)
```

### 3.2.2 Admin based Authentication

We can optionally send service account credential to our app that will allow our server to authenticate with Firebase as an **admin** and disregard any security rules.

#### Service Account Secret File

The following example uses the service account secrets *file* path as the value for *serviceAccount* key.

```
# Import Firebase REST API library
import firebase

# Firebase configuration with service account secret file path
config = {
    "apiKey": "apiKey",
    "authDomain": "projectId.firebaseapp.com",
    "databaseURL": "https://databaseName.firebaseio.com",
    "projectId": "projectId",
    "storageBucket": "projectId.appspot.com",
    "messagingSenderId": "messagingSenderId",
    "appId": "appId"

    "serviceAccount": "path/to/serviceAccountCredentials.json"
}

firebaseApp = firebase.initialize_app(config)
```

#### Service Account Secret Dict

The following example uses the service account secrets *dict* as the value for *serviceAccount* key.

```
# Import Firebase REST API library
import firebase

# Firebase configuration
config = {
    "apiKey": "apiKey",
    "authDomain": "projectId.firebaseapp.com",
    "databaseURL": "https://databaseName.firebaseio.com",
    "projectId": "projectId",
    "storageBucket": "projectId.appspot.com",
```

(continues on next page)

(continued from previous page)

```
    "messagingSenderId": "messagingSenderId",
    "appId": "appId"
}

# Service Account Secret dict
service_account_key = {
    "type": "service_account",
    "project_id": "project_id",
    "private_key_id": "private_key_id",
    "private_key": "private_key",
    "client_email": "client_email",
    "client_id": "client_id",
    "auth_uri": "https://accounts.google.com/o/oauth2/auth",
    "token_uri": "https://oauth2.googleapis.com/token",
    "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
    "client_x509_cert_url": "client_x509_cert_url"
}

config['serviceAccount'] = service_account_key

firebaseApp = firebase.initialize_app(config)
```

**Note:** Adding a service account will authenticate as an admin by default for all database queries, check out the *Authentication documentation* for how to authenticate users.

### 3.2.3 Use Services

A Firebase app can use multiple Firebase services.

`firebaseApp.auth()` - *Authentication*

`firebaseApp.database()` - *Database*

`firebaseApp.storage()` - *Storage*

Check out the documentation for each service for further details.

## 3.3 Authentication

The authentication service allows you to signup, login, edit profile, apply security to the data you might store in either *Database* or *Storage*, and of course delete your account.

```
# Get a reference to the auth service
auth = firebaseApp.auth()
```

**Note:** All sign in methods return user data, including a token you can use to adhere the security rules.

### 3.3.1 create_user_with_email_and_password

Users can create an account using their email address and choice of password.

```
# Creating an account
auth.create_user_with_email_and_password(email, password)
```

> **Note:** Make sure you have the Email/Password provider enabled in your Firebase dashboard under Authentication -> Sign In Method.

### 3.3.2 sign_in_with_email_and_password

User can login using their email and password, provided they *created an account* first.

```
# Log the user in
user = auth.sign_in_with_email_and_password(email, password)
```

### 3.3.3 create_custom_token

You can also create users using custom tokens,
For example:

```
# Create custom token
token = auth.create_custom_token("your_custom_id")
```

You can also pass in additional claims.

```
# Create custom token with claims
token_with_additional_claims = auth.create_custom_token("your_custom_id", {"premium_
↪account": True})
```

> **Note:** You need admin credentials (Service Account Key) to create custom tokens.

### 3.3.4 sign_in_with_custom_token

You can send these custom tokens to the client to sign in, or sign in as the user on the server.

```
# log in user using custom token
user = auth.sign_in_with_custom_token(token)
```

### 3.3.5 sign_in_anonymous

Allows users (who haven't signed up yet) to use your app without creating an account.

```
# Log the user in anonymously
user = auth.sign_in_anonymous()
```

> **Note:** Make sure you have the **Anonymous** provider enabled in your Firebase dashboard under Authentication -> Sign In Method.

### 3.3.6 create_authentication_uri

Signing in with social providers is done through two steps. First step one is done via redirecting user to the providers' login page using *create_authentication_uri* which is can be used dynamically for all providers.

> **Warning:** At the moment only sign is via **Google** is supported, other ones might break or work.

The method returns an link to redirect user to providers' sign in page. Once the user signs into their account, user is asked for permissions and when granted, are redirect to the uri set while creating **OAuth Client IDs**, with authorization code to which can be further used to generate tokens to sign in with social providers in *second step*.

```
# Get a reference to the auth service with provider secret set
auth = firebaseApp.auth(client_secret='client-secret-file.json')
```

```
# Example usage with Flask
@auth.route('/login/google')
def login_google():
   return redirect(auth.create_authentication_uri('google.com'))
```

> **Note:** Make sure you have the **social** provider enabled in your Firebase dashboard under Authentication -> Sign In Method.

#### authenticate_login_with_google

This method is actually an reference to *create_authentication_uri* with **Google** preset as the provider to use.

```
# Example usage with Flask
@auth.route('/login/google')
def login_google():
   return redirect(auth.authenticate_login_with_google())
```

> **Note:** Make sure you have the **Google Sign In** provider enabled in your Firebase dashboard under Authentication -> Sign In Method.

### 3.3.7 sign_in_with_oauth_credential

Second step to sign in using social provider is to pass the URL (containing multiple params) that the user is redirected to, into this method. This method auto generates the tokens using params from that URL, then signs the user in using those tokens to Firebase linking the specific provider.

```
# Here https://example.com/oauth2callback/ is the redirect URI
# that was set while creating OAuth Client ID

# Example usage with Flask
@auth.route('/oauth2callback/')
def oauth2callback():

   user = auth.sign_in_with_oauth_credential(request.url)

        return jsonify(**user)
```

### 3.3.8 get_account_info

This method returns an detailed version of the user's data associated with Authentication service.

```
# User account info
user_info = auth.get_account_info(user['idToken'])
```

### 3.3.9 update_profile

Update stored information or add information into the user's account.

```
# Update user's name
auth.update_profile(user['idToken'], display_name='Iron Man')

# update user's profile picture
auth.update_profile(user['idToken'], photo_url='https://i.pinimg.com/originals/c0/37/2f/
↪c0372feb0069e6289eb938b219e0b0a1.jpg')
```

### 3.3.10 refresh

Firebase Auth Tokens are granted when an user logs in, and are associated with an expiration time of an hour generally, after that they lose validation and a new set of Tokens are needed, and they can be obtained by passing the `refreshToken` key from the users' tokens, received when logged in.

```
# before the 1 hour expiry:
user = auth.refresh(user['refreshToken'])

# now we have a fresh token
user['idToken']
```

### 3.3.11 delete_user_account

In case any user want to delete their account, it can be done by passing `idToken` key from the users' tokens, received when logged in.

```
auth.delete_user_account(user['idToken'])
```

### 3.3.12 send_password_reset_email

In case any user forgot his password, it is possible to send them email containing an code or link to reset their password.

```
auth.send_password_reset_email(email)
```

### 3.3.13 send_email_verification

To ensure the email address belongs to the user who created the account, it is recommended to request verification of the email. Verification code/link can be sent to the user by passing `idToken` key from the users' tokens, to this method.

```
auth.send_email_verification(user['idToken'])
```

## 3.4 Database

The database service allows you to run CRUD operations to your Firebase Realtime Database, and also perform complex queries while doing so.

```
# Create database instance
db = firebaseApp.database()
```

> **Note:** Each of the following methods accepts a user token: *get()*, *push()*, *set()*, *update()*, *remove()* and *stream()*.

### 3.4.1 Build Path

You can build paths to your data by using the `child()` method.

```
db.child("users").child("Edward")

# Alternate ways
db.child("users", "Edward")
db.child("users/Edward")
```

### 3.4.2 Save Data

#### push

To save data with a unique, auto-generated, timestamp-based key, use the `push()` method.

```
data = {"name": "Anthony 'Edward' Stark"}
db.child("users").push(data)
```

#### set

To create your own keys use the `set()` method. The key in the example below is "Morty".

```
data = {"name": "Anthony 'Edward' Stark"}
db.child("users").child("Edward").set(data)
```

#### update

To update data for an existing entry use the `update()` method.

```
db.child("users").child("Edward").update({"name": "Tony Stark"})
```

#### remove

To delete data for an existing entry use the `remove()` method.

```
db.child("users").child("Edward").remove()
```

#### multi-location updates

You can also perform multi-location updates with the `update()` method.

```
data = {
    "users/Edward/": {
        "name": "Anthony 'Edward' Stark"
    },
    "users/Pepper/": {
        "name": "Virginia 'Pepper' Potts"
    }
}

db.update(data)
```

To perform multi-location writes to new locations we can use the `generate_key()` method.

```
data = {
    "users/"+ref.generate_key(): {
        "name": "Anthony 'Edward' Stark"
    },
```

(continues on next page)

(continued from previous page)

```
    "users/"+ref.generate_key(): {
        "name": "Virginia 'Pepper' Potts"
    }
}

db.update(data)
```

### 3.4.3 Retrieve Data

#### get

To return data from a path simply call the `get()` method.

```
users = db.child("users").get()
```

#### each

Returns a list of objects on each of which you can call `val()` and `key()`.

```
users = db.child("users").get()
for user in users.each():
    print(user.key(), user.val())

# Output:
# Edward {name": "Anthony 'Edward' Stark"}
# Pepper {'name': "Virginia 'Pepper' Potts"}
```

#### val

Queries return a PyreResponse object. Calling `val()` on these objects returns the query data.

```
users = db.child('users').child('Edward').get()

for user in users.each():
   print(user.val())

# Output:
# {'name': "Anthony 'Edward' Stark"}
```

### key

Calling `key()` returns the key for the query data.

```
users = db.child("users").get()

for user in users.each():
   print(user.key())

# Output:
# Edward
# Pepper
```

### Conditional Requests

It's possible to do conditional sets and removes by using the `conditional_set()` and `conitional_remove()` methods respectively. You can read more about conditional requests in Firebase here.

To use these methods, you first get the ETag of a particular path by using the `get_etag()` method. You can then use that tag in your conditional request.

```
etag = db.child("users").child("Edward").get_etag()
data = {"name": "Tony Stark"}
db.child("users").child("Edward").conditional_set(data, etag)
```

If the passed ETag does not match the ETag of the path in the database, the data will not be written, and both conditional request methods will return a single key-value pair with the new ETag to use of the following form:

```
{ "ETag": "8KnE63B6HiKp67Wf3HQrXanujSM=" }
```

Here's an example of checking whether or not a conditional removal was successful:

```
etag = db.child("users").child("Edward").get_etag()
response = db.child("users").child("Edward").conditional_remove(etag)

if "ETag" in response:
    etag = response["ETag"] # our ETag was out-of-date
else:
    print("We removed the data successfully!")
```

### shallow

To return just the keys at a particular path use the `shallow()` method.

```
all_user_ids = db.child("users").shallow().get()
```

> **Note:** `shallow()` can not be used in conjunction with any complex queries.

#### streaming

You can listen to live changes to your data with the `stream()` method.

```
def stream_handler(message):
    print(message["event"]) # put
    print(message["path"]) # /-K7yGTTEp7O549EzTYtI
    print(message["data"]) # {'title': 'Firebase', "body": "etc..."}

my_stream = db.child("posts").stream(stream_handler)
```

You should at least handle `put` and `patch` events. Refer to "Streaming from the REST API" for details.

You can also add a `stream_id` to help you identify a stream if you have multiple running:

```
my_stream = db.child("posts").stream(stream_handler, stream_id="new_posts")
```

#### close the stream

```
my_stream.close()
```

### 3.4.4 Complex Queries

Queries can be built by chaining multiple query parameters together.

```
users_by_name = db.child("users").order_by_child("name").limit_to_first(3).get()
```

This query will return the first three users ordered by name.

#### order_by_child

We begin any complex query with `order_by_child()`.

```
users_by_name = db.child("users").order_by_child("name").get()
```

This query will return users ordered by name.

#### equal_to

Return data with a specific value.

```
users_by_score = db.child("users").order_by_child("score").equal_to(10).get()
```

This query will return users with a score of 10.

#### start_at and end_at

Specify a range in your data.

```
users_by_score = db.child("users").order_by_child("score").start_at(3).end_at(10).get()
```

This query returns users ordered by score and with a score between 3 and 10.

#### limit_to_first and limit_to_last

Limits data returned.

```
users_by_score = db.child("users").order_by_child("score").limit_to_first(5).get()
```

This query returns the first five users ordered by score.

#### order_by_key

When using `order_by_key()` to sort your data, data is returned in ascending order by key.

```
users_by_key = db.child("users").order_by_key().get()
```

#### order_by_value

When using `order_by_value()`, children are ordered by their value.

```
users_by_value = db.child("users").order_by_value().get()
```

### 3.4.5 Helper Methods

#### generate_key

`db.generate_key()` is an implementation of Firebase's key generation algorithm.

See *multi-location updates* for a potential use case.

#### sort

Sometimes we might want to sort our data multiple times. For example, we might want to retrieve all articles written between a certain date then sort those articles based on the number of likes.

Currently the REST API only allows us to sort our data once, so the `sort()` method bridges this gap.

```
articles = db.child("articles").order_by_child("date").start_at(startDate).end_
↪at(endDate).get()
articles_by_likes = db.sort(articles, "likes")
```

### 3.4.6 Common Errors

#### Index not defined

- Indexing is **not enabled** for the database reference.

## 3.5 Storage

The storage service allows you to upload files (eg. text, image, video) to Firebase Storage.

```
# Create storage instance
storage = firebaseApp.storage()
```

### 3.5.1 child

Just like with the Database service, you can build paths to your data with the Storage service.

```
storage.child("images/example.jpg")

# Alternative ways
storage.child("images").child("example.jpg")
storage.child("images", "example.jpg")
```

### 3.5.2 put

The put method takes the path to the local file and an optional user token.

```
# as admin
storage.child("images/example.jpg").put("example2.jpg")

# as user
storage.child("images/example.jpg").put("example2.jpg", user['idToken'])
```

### 3.5.3 download

The download method takes the path to the saved database file and the name you want the downloaded file to have.

```
# as admin
 storage.child("images/example.jpg").download("downloaded.jpg")

 # as user
 storage.child("images/example.jpg").download("downloaded.jpg", user['idToken'])
```

### 3.5.4 get_url

The get_url method takes the path to the saved database file and user token which returns the storage url.

```
# as admin
storage.child("images/example.jpg").get_url()

# as admin with expiration time for link to expire
storage.child("images/example.jpg").get_url(expiration_hour=12)

# as user
storage.child("images/example.jpg").get_url(user["idToken"])

# returned URL example:
# https://firebasestorage.googleapis.com/v0/b/storage-url.appspot.com/o/images%2Fexample.
↪jpg?alt=media&token=$token
```

### 3.5.5 delete

The delete method takes the path to the saved database file and user token.

```
# as admin
storage.child("images/example.jpg").delete()

# as user
storage.child("images/example.jpg").delete(user["idToken"])
```

### 3.5.6 list_of_files

The list_of_files method works only if used under admin credentials.

```
# as admin
storage.list_of_files()
```

## 3.6 API Reference

### 3.6.1 firebase

A simple python wrapper for Google's Firebase REST APIs.

**class** firebase.**Firebase**(*config*)

Bases: object

Firebase Interface

**Parameters**
**config** (*dict*) – Firebase configuration

**auth**(*client_secret=None*)

Initializes and returns a new Firebase Authentication instance.

**Parameters**
**client_secret** (*str or dict*) – (Optional) File path to or the dict object from social client secret file, defaults to None.

**Returns**
A newly initialized instance of Auth.

**Return type**
*Auth*

**database**()

Initializes and returns a new Firebase Realtime Database instance.

**Returns**
A newly initialized instance of Database.

**Return type**
*Database*

**storage**()

Initializes and returns a new Firebase Storage instance.

**Returns**
A newly initialized instance of Storage.

**Return type**
*Storage*

firebase.**initialize_app**(*config*)

Initializes and returns a new Firebase instance.

**Parameters**
**config** (*dict*) – Firebase configuration

**Returns**
A newly initialized instance of Firebase.

**Return type**
*Firebase*

### 3.6.2 firebase.auth package

A simple python wrapper for Google's Firebase Authentication REST API

**class** firebase.auth.**Auth**(*api_key*, *credentials*, *requests*, *client_secret=None*)

Bases: object

Firebase Authentication Service

**Parameters**

- **api_key** (*str*) – apiKey from Firebase configuration
- **credentials** (Credentials) – Service Account Credentials
- **requests** (Session) – Session to make HTTP requests
- **client_secret** (*str or dict*) – (Optional) File path to or the dict object from social client secret file, defaults to None.

`authenticate_login_with_google()`

Redirect the user to Google's OAuth 2.0 server to initiate the authentication and authorization process.

**Returns**
Google Sign In URL

**Return type**
str

`create_authentication_uri(provider_id)`

Creates an authentication URI for the given social provider.

For more details:
Firebase Auth REST API | Fetch providers for email

**Parameters**
**provider_id** (*str*) – The IdP ID. For white listed IdPs it's a short domain name e.g. 'google.com', 'aol.com', 'live.net' and 'yahoo.com'. For other OpenID IdPs it's the OP identifier.

**Returns**
The URI used by the IDP to authenticate the user.

**Return type**
str

`create_custom_token(uid, additional_claims=None, expiry_minutes=60)`

Create a Firebase Auth custom token.

For more details:
Firebase Documentation | Create Custom tokens

**Parameters**

- **uid** (*str*) – The unique identifier of the user, must be a string, between 1-36 characters long.
- **additional_claims** (*dict or None*) – Optional custom claims to include in the Security Rules `auth` / `request.auth` variables.
- **expiry_minutes** (*int*) – The time, in minutes since the UNIX epoch, at which the token expires. Default value is 60.

**Returns**
Firebase Auth custom token.

**Return type**
str

`create_user_with_email_and_password(email, password)`

Create a new user with email and password.

For more details:
Firebase Auth REST API | section-create-email-password

**Parameters**

- **email** (*str*) – The email for the user to create.
- **password** (*str*) – The password for the user to create.

**Returns**
User Email and Firebase Auth Tokens.

**Return type**
dict

**delete_user_account**(*id_token*)
Delete an existing user.

For more details:
Firebase Auth REST API | section-delete-account

**Parameters**
**id_token** (*str*) – The Firebase ID token of the user to delete.

**get_account_info**(*id_token*)
Fetch user's stored account information.

For more details:
Firebase Auth REST API | section-get-account-info

**Parameters**
**id_token** (*str*) – The Firebase ID token of the account.

**Returns**
The account info, associated with the given Firebase ID token.

**Return type**
dict

**refresh**(*refresh_token*)
Refresh a Firebase ID token.

For more details:
Firebase Auth REST API | section-refresh-token

**Parameters**
**refresh_token** (*str*) – A Firebase Auth refresh token.

**Returns**
New (Refreshed) Firebase Auth tokens for the account.

**Return type**
dict

`send_email_verification`(*id_token*)

Send an email verification to verify email ownership.

For more details:
Firebase Auth REST API | section-send-email-verification

**Parameters**
**id_token** (*str*) – The Firebase ID token of the user to verify.

**Returns**
The email of the account associated with Firebase ID token.

**Return type**
dict

`send_password_reset_email`(*email*)

Send a password reset email.

For more details:
Firebase Auth REST API | section-send-password-reset-email

**Parameters**
**email** (*str*) – User's email address.

**Returns**
User's email address.

**Return type**
dict

`sign_in_anonymous`()

Sign In Anonymously.

For more details:
Firebase Auth REST API | section-sign-in-anonymously

**Returns**
Firebase Auth Tokens.

**Return type**
dict

`sign_in_with_custom_token`(*token*)

Exchange custom token for an ID and refresh token.

For more details:
Firebase Auth REST API | section-verify-custom-token

**Parameters**
**token** (*str*) – A Firebase Auth custom token from which to create an ID and refresh token pair.

**Returns**
Firebase Auth Tokens.

**Return type**
dict

**sign_in_with_email_and_password**(*email*, *password*)
Sign in a user with an email and password.

For more details:
Firebase Auth REST API | section-sign-in-email-password

**Parameters**
- **email** (*str*) – The email the user is signing in with.
- **password** (*str*) – The password for the account.

**Returns**
UserInfo and Firebase Auth Tokens.

**Return type**
dict

**sign_in_with_oauth_credential**(*oauth2callback_url*)
Sign In With OAuth credential.

For more details:
Firebase Auth REST API | Sign in with OAuth credential

**Parameters**
**oauth2callback_url** (*str*) – The URL redirected to after successful authorization from the provider.

**Returns**
User account info and Firebase Auth Tokens.

**Return type**
dict

**update_profile**(*id_token*, *display_name=None*, *photo_url=None*, *delete_attribute=None*)
Update a user's profile (display name / photo URL).

For more details:
Firebase Auth REST API | section-update-profile

**Parameters**
- **id_token** (*str*) – A Firebase Auth ID token for the user.

- **display_name** (*str or None*) – User's new display name.
- **photo_url** (*None or str*) – User's new photo url.
- **delete_attribute** (*list[str] or None*) – List of attributes to delete, "DISPLAY_NAME" or "PHOTO_URL". This will nullify these values.

**Returns**
UserInfo and Firebase Auth Tokens.

**Return type**
dict

**verify_password_reset_code**(*reset_code*, *new_password*)

Reset password using code.

For more details:
Firebase Auth REST API | #section-confirm-reset-password

**Parameters**

- **reset_code** (*str*) – The email action code sent to the user's email for resetting the password.
- **new_password** (*str*) – The user's new password.

**Returns**
User Email and Type of the email action code.

**Return type**
dict

### 3.6.3 firebase.database package

A simple python wrapper for Google's Firebase Database REST API

**class** firebase.database.**Database**(*credentials*, *database_url*, *requests*)

Bases: object

Firebase Database Service

**Parameters**

- **credentials** (Credentials) – Service Account Credentials.
- **database_url** (*str*) – databaseURL from Firebase configuration.
- **requests** (Session) – Session to make HTTP requests.

**build_headers**(*token=None*)

Build Request Header.

**Parameters**
**token** (*str*) – (Optional) Firebase Auth User ID Token, defaults to None.

**Returns**
Request Header.

**Return type**
dict

`build_request_url`(*token*)

Builds Request URL for query.

**Parameters**
**token** (*str*) – Firebase Auth User ID Token

**Returns**
Request URL

**Return type**
str

`check_token`(*database_url*, *path*, *token*)

Builds Request URL to write/update/remove data.

**Parameters**

- **database_url** (*str*) – databaseURL from Firebase configuration.
- **path** (*str*) – Path to data.
- **token** (*str*) – Firebase Auth User ID Token

**Returns**
Request URL

**Return type**
str

`child`(**args*)

Build paths to your data.

**Parameters**
**args** (*str*) – Positional arguments to build path to database.

**Returns**
A reference to the instance object.

**Return type**
*Database*

`conditional_remove`(*etag*, *token=None*)

Conditionally delete data from database.

For more details:
Firebase Database REST API | Conditional Requests | section-expected-responses

**Parameters**

- **etag** (*str*) – Unique identifier for specific data at a specified location.
- **token** (*str*) – (Optional) Firebase Auth User ID Token, defaults to `None`.

**Returns**
Successful attempt returns `None`, in case of ETag mismatch an updated ETag for the specific data is returned in `dict` object

**Return type**
`None`

**conditional_set**(*data*, *etag*, *token=None*, *json_kwargs={}*)

Conditionally add data to database.

For more details:
Firebase Database REST API | Conditional Requests | section-expected-responses

**Parameters**

- **data** (*dict*) – Data to be stored in database.
- **etag** (*str*) – Unique identifier for specific data at a specified location.
- **token** (*str*) – (Optional) Firebase Auth User ID Token, defaults to `None`.
- **json_kwargs** (*dict*) – (Optional) Keyword arguments to send to `json.dumps()` methods for serialization of data, defaults to `{}` (empty `dict` object).

**Returns**
Successful attempt returns the data specified to store, failed attempt (due to ETag mismatch) returns the current `ETag` for the specified path.

**Return type**
dict

**end_at**(*end*)

Filter data where child key value ends at specified value.

For more details:
Firebase Documentation | Retrieve Data | Complex Filtering | range-queries

**Parameters**
**end** (*int or float or str*) – Arbitrary ending points for queries.

**Returns**
A reference to the instance object.

**Return type**
*Database*

**equal_to**(*equal*)

Filter data where child key value is equal to specified value.

For more details:
Firebase Documentation | Retrieve Data | Complex Filtering | range-queries

**Parameters**
**equal** (*int or float or str*) – Arbitrary point for queries.

**Returns**
A reference to the instance object.

**Return type**
*Database*

`generate_key()`

Generate Firebase's push IDs.

For more details:
Firebase Blog | The 2^120 Ways to Ensure Unique Identifiers

**Returns**
Firebase's push IDs

**Return type**
str

`get(token=None, json_kwargs={})`

Read data from database.

For more details:
Firebase Database REST API | GET - Reading Data

**Parameters**

- **token** (*str*) – (Optional) Firebase Auth User ID Token, defaults to `None`.
- **json_kwargs** (*dict*) – (Optional) Keyword arguments to send to `json.dumps()` method for deserialization of data, defaults to `{}` (empty `dict` object).

**Returns**
The data associated with the path.

**Return type**
dict

`get_etag(token=None)`

Fetches Firebase ETag at a specified location.

For more details:
Firebase Database REST API | Conditional Requests | #section-cond-etag

**Parameters**
**token** (*str*) – (Optional) Firebase Auth User ID Token, defaults to `None`.

**Returns**
Firebase ETag

**Return type**
str

`limit_to_first(limit_first)`

Filter the number of data to receive from top.

For more details:

Firebase Documentation | Retrieve Data | Complex Filtering | limit-queries

**Parameters**
**limit_first** (*int*) – Maximum number of children to select from top.

**Returns**
A reference to the instance object.

**Return type**
*Database*

**limit_to_last**(*limit_last*)

Filter the number of data to receive from bottom.

For more details:
Firebase Documentation | Retrieve Data | Complex Filtering | limit-queries

**Parameters**
**limit_last** (*int*) – Maximum number of children to select from bottom.

**Returns**
A reference to the instance object.

**Return type**
*Database*

**order_by_child**(*order*)

Filter data by a common child key.

For more details:
Firebase Documentation | Retrieve Data | Filtering Data | filtering-by-a-specified-child-key

**Parameters**
**order** (*str*) – Child key name.

**Returns**
A reference to the instance object.

**Return type**
*Database*

**order_by_key**()

Filter data by their keys.

For more details:
Firebase Documentation | Retrieve Data | Filtering Data | filtering_by_key

**Returns**
A reference to the instance object.

**Return type**
*Database*

`order_by_value`()

Filter data by the value of their child keys.

For more details:
Firebase Documentation | Retrieve Data | Filtering Data | filtering-by-value

**Returns**
A reference to the instance object.

**Return type**
*Database*

`push`(*data*, *token=None*, *json_kwargs={}*)

Add data to database.

This method adds a Firebase Push ID at the end of the specified path, and then adds/stores the data in database, unlike *set()* which does not use a Firebase Push ID.

For more details:
Firebase Database REST API | POST - Pushing Data

**Parameters**

- **data** (*dict*) – Data to be stored in database.
- **token** (*str*) – (Optional) Firebase Auth User ID Token, defaults to None.
- **json_kwargs** (*dict*) – (Optional) Keyword arguments to send to json.dumps() method for serialization of data, defaults to {} (empty dict object).

**Returns**
Child key (Firebase Push ID) name of the data.

**Return type**
dict

`remove`(*token=None*)

Delete data from database.

For more details:
Firebase Database REST API | DELETE - Removing Data

**Parameters**
**token** (*str*) – (Optional) Firebase Auth User ID Token, defaults to None.

**Returns**
Successful attempt returns None.

**Return type**
None

**set**(*data*, *token=None*, *json_kwargs={}*)

Add data to database.

This method writes the data in database in the specified path, unlike *push()* which creates a Firebase Push ID then writes the data to database.

For more details:
Firebase Database REST API | PUT - Writing Data

**Parameters**

- **data** (*dict*) – Data to be stored in database.
- **token** (*str*) – (Optional) Firebase Auth User ID Token, defaults to None.
- **json_kwargs** (*dict*) – (Optional) Keyword arguments to send to json.dumps() method for serialization of data, defaults to {} (empty dict object).

**Returns**
Successful attempt returns the data specified to add to database.

**Return type**
dict

**shallow**()

Limit the depth of the response.

For more details:
Firebase Database REST API | Query Parameters | section-param-shallow |

**Returns**
A reference to the instance object.

**Return type**
*Database*

**sort**(*origin*, *by_key*, *reverse=False*)

Further sort data based on a child key value.

**Parameters**

- **origin** (*dict*) – Data to be sorted (generally the output from *get()* method).
- **by_key** (*str*) – Child key name to sort by.
- **reverse** (*bool*) – (Optional) Whether to return data in descending order, defaults to False (data is returned in ascending order).

**Returns**
Sorted version of the data.

**Return type**
dict

**start_at**(*start*)

Filter data where child key value starts from specified value.

For more details:
Firebase Documentation | Retrieve Data | Complex Filtering | range-queries

**Parameters**
**start** (*int or float or str*) – Arbitrary starting points for queries.

**Returns**
A reference to the instance object.

**Return type**
*Database*

**stream**(*stream_handler*, *token=None*, *stream_id=None*, *is_async=True*)

**update**(*data*, *token=None*, *json_kwargs={}*)

Update stored data of database.

For more details:
Firebase Database REST API | PATCH - Updating Data

**Parameters**

- **data** (*dict*) – Data to be updated.
- **token** (*str*) – (Optional) Firebase Auth User ID Token, defaults to `None`.
- **json_kwargs** (*dict*) – (Optional) Keyword arguments to send to `json.dumps()` method for serialization of data, defaults to `{}` (empty `dict` object).

**Returns**
Successful attempt returns the data specified to update.

**Return type**
dict

### 3.6.4 firebase.storage package

A simple python wrapper for Google's Firebase Cloud Storage REST API

**class** firebase.storage.**Storage**(*credentials*, *requests*, *storage_bucket*)

Bases: `object`

Firebase Cloud Storage Service

**Parameters**

- **credentials** (`Credentials`) – Service Account Credentials.
- **requests** (`Session`) – Session to make HTTP requests.
- **storage_bucket** (*str*) – `storageBucket` from Firebase configuration.

**child**(**args*)

Build paths to your storage.

**Parameters**
**args** (*str*) – Positional arguments to build path to storage.

**Returns**
A reference to the instance object.

**Return type**
*Storage*

**delete**(*token=None*)

Delete file from storage.

For more details:
Firebase Documentation | Delete files with Cloud Storage on Web

**Parameters**
**token** (*str*) – (Optional) Firebase Auth User ID Token, defaults to None.

**download**(*filename*, *token=None*)

Download file from storage.

For more details:
Firebase Documentation | Download files with Cloud Storage on Web

**Parameters**

- **filename** (*str*) – File name to be downloaded as.
- **token** (*str*) – (Optional) Firebase Auth User ID Token, defaults to None.

**get_url**(*token=None*, *expiration_hour=24*)

Fetches URL for file.

**Parameters**

- **token** (*str*) – (Optional) Firebase Auth User ID Token, defaults to None.
- **expiration_hour** (*int*) – (Optional) time in hour for URL to expire after, defaults to 24 hours. Works only for links generated with admin credentials.

**Returns**
URL for the file.

**Return type**
str

**list_files**()

List of all files in storage.

for more details:
Firebase Documentation | List files with Cloud Storage on Web

**Returns**
list of `Blob`

**Return type**
`_BlobIterator`

`put`(*file*, *token=None*)

Upload file to storage.

For more details:
Firebase Documentation | Upload files with Cloud Storage on Web

**Parameters**

- **file** (*str*) – Local path to file to upload.
- **token** (*str*) – (Optional) Firebase Auth User ID Token, defaults to `None`.

**Returns**
Successful attempt returns `None`.

**Return type**
`None`

CHAPTER

FOUR

# INDICES AND TABLES

- genindex
- modindex

# PYTHON MODULE INDEX

## f

# INDEX

## P



## R



## S



## U



## V