HELP DOCUMENTS FOR FIREBASE REALTIME DATABASE

Introduction to the Admin Database API

With the Admin SDK, you can read and write Realtime Database data with full admin privileges, or with finer-grained limited privileges. In this document, we'll guide you through adding the Firebase Admin SDK to your project for accessing the Firebase Realtime Database.

Admin SDK Setup

To get started with the Firebase Realtime Database on your server, you'll first need to set up the Firebase Admin SDK in your language of choice

If you are interested in using the Node.js SDK as a client for end-user access (for example, in a Node.js desktop or IoT 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.

Admin SDK Authentication

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:

Firebase Admin SDK Auth Access Levels	
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.

Authenticate with admin privileges

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

FOR PYTHON:

```python
import firebase_admin
from firebase_admin import credentials
from firebase_admin import db

# Fetch the service account key JSON file contents
cred = credentials.Certificate('path/to/serviceAccountKey.json')

# Initialize the app with a service account, granting admin privileges
firebase_admin.initialize_app(cred, {
    'databaseURL': 'https://databaseName.firebaseio.com'
})

# As an admin, the app has access to read and write all data, regradless of Security Rules
ref = db.reference('restricted_access/secret_document')
print(ref.get())
```

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

Authenticate with limited privileges

As a best practice, a service should have access to only the resources it needs. To get more fine-grained control over the resources a Firebase app instance can access, use a unique identifier in your Security Rules to represent your service. Then set up appropriate rules which grant your service access to the resources it needs. For example:

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

Then, on your server, when you initialize the Firebase app, use the `databaseAuthVariableOverride` option 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.

FOR PYTHON:

```python
import firebase_admin
from firebase_admin import credentials
from firebase_admin import db

# Fetch the service account key JSON file contents
cred = credentials.Certificate('path/to/serviceAccountKey.json')

# Initialize the app with a custom auth variable, limiting the server's access
firebase_admin.initialize_app(cred, {
    'databaseURL': 'https://databaseName.firebaseio.com',
    'databaseAuthVariableOverride': {
        'uid': 'my-service-worker'
    }
})

# The app only has access as defined in the Security Rules
ref = db.reference('/some_resource')
print(ref.get())
```

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.

FOR PYTHON:

```python
import firebase_admin
from firebase_admin import credentials
from firebase_admin import db

# Fetch the service account key JSON file contents
cred = credentials.Certificate('path/to/serviceAccountKey.json')

# Initialize the app with a None auth variable, limiting the server's access
firebase_admin.initialize_app(cred, {
    'databaseURL': 'https://databaseName.firebaseio.com',
    'databaseAuthVariableOverride': None
})

# The app only has access to public data as defined in the Security Rules
ref = db.reference('/public_resource')
print(ref.get())
```

FOR MORE INFORMATION:

Please visit the https://firebase.google.com/docs/database/admin to set your realtime database, and please visit the https://firebase.google.com/docs/admin/setup#add_firebase_to_your_app to add the Firebase Admin SDK to your server.

REFERENCES FROM **GOOGLE FIREBASE DOCUMENTATION**