Unity Firebase Rest API

Firebase REST API Implementation for Unity.

Requirements

- Newtonsoft JSON Unity Package

Configuration

- Go to the **Assets > FirebaseRestAPI > Resources > FirebaseConfig** scriptable object.
- Enter your Firebase project's **"Id”** and **"Web Api Key"**.
- To save the **Firebase Auth** session locally, enable the **"Save Session"** option and assign a **Session Saver**. You can either use the **DefaultSessionSaver** or create your own save logic by extending the **SessionSaver** scriptable object class.
- Go to **DefaultSessionSaver** scriptable object and paste your **AES-128-bit** key for encrypting the auth session. You can access the **Encryption Key Generator** tool through the **DefaultSessionSaver's** inspector or by clicking **Assets > FirebaseRestAPI > Encryption Key Generator**.

Authentication

Sign up with email and password

```
FirebaseAuth.SignUpWithEmailAndPassword(email, password, (user) =>
{
    Debug.Log("User signed up");

}, (error) =>
{
    if (error.message == AuthError.EMAIL_EXISTS)
    {
```

```
        Debug.Log("Email already exists");
    }
    else if (error.message == AuthError.OPERATION_NOT_ALLOWED)
    {
        Debug.Log("Password sign-in is disabled for this project.");
    }
    else if (error.message == AuthError.WEAK_PASSWORD)
    {
        Debug.Log("Weak password");
    }
    else
    {
        Debug.Log($"Signup failed: {error.message}");
    }
});
```

Sign in with email and password

```
FirebaseAuth.SignInWithEmailAndPassword(email, password, (user) =>
{
    Debug.Log("User signed in");

}, (error) =>
{
    if (error.message == AuthError.EMAIL_NOT_FOUND)
    {
       Debug.Log("Email not found");
    }
    else if (error.message == AuthError.INVALID_PASSWORD)
    {
        Debug.Log("Invalid password");
    }
    else if (error.message == AuthError.USER_DISABLED)
    {
        Debug.Log("User disabled");
    }
    else
    {
        Debug.Log($"Signin failed: {error.message}");
    }
});
```

Sign in with OAuth credential

```
FirebaseAuth.SignInWithOAuthCredential(googleIdToken, new GoogleAuthProvider(), (user) =>
{
    Debug.Log("User signed in with Google");

}, (error) =>
{
    if (error.message == AuthError.INVALID_IDP_RESPONSE)
    {
        Debug.Log("Invalid IDP response");
    }
    else if (error.message == AuthError.OPERATION_NOT_ALLOWED)
    {
        Debug.Log("Google sign-in is disabled for this project.");
    }
```

```
    else
    {
        Debug.Log($"Google Signin failed: {error.message}");
    }
});
```

Get user data

Some user data, such as photo URL and display name, is not retrieved automatically after sign-in. You can call **FirebaseAuth.GetUserData()** to retrieve the additional user data. After a successful response, the user data will be accessible through **FirebaseAuth.CurrentUser.UserData**.

```
FirebaseAuth.GetUserData((userData) =>
{
    Debug.Log("User data received");
}, (error) =>
{
    if (error.message == AuthError.INVALID_ID_TOKEN)
    {
        Debug.Log("Invalid id token");
    }
    else if (error.message == AuthError.USER_NOT_FOUND)
    {
        Debug.Log("User not found");
    }
    else
    {
        Debug.Log($"Get user data failed: {error.message}");
    }
});
```

Reset password

```
FirebaseAuth.SendPasswordResetEmail(email, () =>
{
    Debug.Log("Password reset email sent");

}, (error) =>
{
    if (error.message == AuthError.EMAIL_NOT_FOUND)
    {
        Debug.Log("Email not found");
    }
    else
    {
        Debug.Log($"Password reset failed: {error.message}");
    }
});
```

Change password

```
FirebaseAuth.ChangePassword(newPassword, () =>
{
    Debug.Log("Password changed successfully");
}, (error) =>
{
    if (error.message == AuthError.INVALID_ID_TOKEN)
    {
```

```
        Debug.Log("Invalid id token");
    }
    else if (error.message == AuthError.WEAK_PASSWORD)
    {
        Debug.Log("Weak password");
    }
    else
    {
        Debug.Log($"Change password failed: {error.message}");
    }
});
```

Update user profile

For any fields you don't want to update (display name or photo URL), pass an empty string or null.

```
FirebaseAuth.UpdateProfile(displayName, photoUrl, () =>
{
    Debug.Log("Profile updated successfully");
}, (error) =>
{
    if (error.message == AuthError.INVALID_ID_TOKEN)
    {
        Debug.Log("Invalid id token");
    }
    else
    {
        Debug.Log($"Update profile failed: {error.message}");
    }
});
```

Refreshing auth token

Firebase auth tokens expire after 1 hour. To handle token updates automatically, the **AuthTokenRefresher** singleton checks for expired tokens when the app starts and then every minute, refreshing them as needed. You can also manually refresh the auth token by calling **FirebaseAuth.RefreshIdToken()**.

Sign out user

```
FirebaseAuth.SignOut();
```

Realtime Database

Reading data

When reading data from the Realtime Database, the response will be a raw JSON string that needs to be deserialized using `JsonConvert.DeserializeObject` to convert it into your desired data type.

```
new DatabaseReference()
    .Child(child)
    .Get()
    .OnSuccess((response) =>
    {
        Debug.Log("Data fetched successfully");
        object data = JsonConvert.DeserializeObject(response);
    })
    .OnError((error) =>
    {
```

```
        Debug.Log($"Data fetch failed : {error.error}");
    });
```

Writing data

When writing data, the value parameter can be of any type - it will be automatically converted to JSON format before sending the request to Firebase.

```
new DatabaseReference()
        .Child(child)
    .Set(value)
    .OnSuccess((response) =>
    {
        Debug.Log("Data set successfully");
    })
    .OnError((error) =>
    {
        Debug.Log($"Data set failed : {error.error}");
    });
```

Pushing data

```
new DatabaseReference(path)
    .Push(value)
    .OnSuccess((childName) =>
    {
        Debug.Log($"Data pushed successfully : {childName}");
    })
    .OnError((error) =>
    {
        Debug.Log($"Data push failed : {error.error}");
    });
```

Removing data

```
new DatabaseReference()
    .Child(child)
    .Delete()
    .OnSuccess((response) =>
    {
        Debug.Log("Data deleted successfully");
    })
    .OnError((error) =>
    {
        Debug.Log($"Data delete failed : {error.error}");
    });
```

Filtering data

Use the query parameters **orderBy**, **startAt**, **endAt**, **equalTo**, **limitToFirst**, and **limitToLast** to filter database queries.

Before using filters, you need to index the relevant keys in the Firebase console. For details, see Index Your Data.

```
// Get first 10 users by ordering them by their age
new DatabaseReference("users")
    .OrderByChild("age")
    .LimitToFirst(10)
```

```
    .Get()
    .OnSuccess((data) =>
    {
        Debug.Log("Data fetched successfully");
        var users = JsonConvert.DeserializeObject<Dictionary<string, User>>(data);
    })
    .OnError((error) =>
    {
        Debug.Log($"Data fetch failed : {error.error}");
    });


// Get all users who live in New York
new DatabaseReference("users")
    .OrderByChild("city")
    .EqualTo("New York")
    .Get()
    .OnSuccess((data) =>
    {
        Debug.Log("Data fetched successfully");
        var users = JsonConvert.DeserializeObject<Dictionary<string, User>>(data);
    })
    .OnError((error) =>
    {
        Debug.Log($"Data fetch failed : {error.error}");
    });


// Get all users whose score is between 100 and 200
new DatabaseReference("users")
    .OrderByChild("score")
    .StartAt(100)
    .EndAt(200)
    .Get()
    .OnSuccess((data) =>
    {
        Debug.Log("Data fetched successfully");
        var users = JsonConvert.DeserializeObject<Dictionary<string, User>>(data);
    })
    .OnError((error) =>
    {
        Debug.Log($"Data fetch failed : {error.error}");
    });
```

Server values

Use **ServerValue.TIMESTAMP** to set the server time in milliseconds, and **ServerValue.Increment()** to increment a field's value.

```
new DatabaseReference("childName")
    .Set(ServerValue.TIMESTAMP)
    .OnSuccess((response) =>
    {
        Debug.Log(response);
    })
    .OnError((error) =>
    {
        Debug.Log(error.Error);
    });
```

```
new DatabaseReference("childName")
    .Set(ServerValue.Increment(5))
    .OnSuccess((response) =>
    {
        Debug.Log(response);
    })
    .OnError((error) =>
    {
        Debug.Log(error.Error);
    });
```

Firestore

Reading document

```
new FirestoreReference()
    .Collection(collection)
    .Document(document)
    .Get()
    .OnSuccess((response) =>
    {
        Debug.Log($"Document fetched successfully : {JsonConvert.SerializeObject(response
    })
    .OnError((error) =>
    {
        Debug.Log("Error: " + error.message);
    });
```

Writing document

```
new FirestoreReference()
    .Collection(collection)
    .Document(document)
    .Set(data)
    .OnSuccess((response) =>
    {
        Debug.Log("Document set successfully");
    })
    .OnError((error) =>
    {
        Debug.Log("Error: " + error.message);
    });
```

Pushing document

```
new FirestoreReference()
    .Collection(collection)
    .Push(data)
    .OnSuccess((documentId) =>
    {
        Debug.Log($"Document pushed successfully : {documentId}");
    })
    .OnError((error) =>
    {
```

```
        Debug.Log("Error: " + error.message);
    });
```

Deleting document

```
new FirestoreReference()
    .Collection(collection)
    .Document(document)
    .Delete()
    .OnSuccess((response) =>
    {
        Debug.Log("Document deleted successfully");
    })
    .OnError((error) =>
    {
        Debug.Log("Error: " + error.message);
    });
```

Running queries

You can query collections using **Where** and **OrderBy** parameters. For compound queries, you'll need to create indexes in the Firebase console. For details, see Manage indexes in Cloud Firestore.

```
// Query users collection where age is greater than 20
new FirestoreReference()
    .Collection("users")
    .Where("age", FieldOperator.GREATER_THAN, 20)
    .RunQuery()
    .OnSuccess((documents) =>
    {
        Debug.Log($"Query fetched successfully : {JsonConvert.SerializeObject(documents)}
    })
    .OnError((error) =>
    {
        Debug.Log("Error: " + error.message);
    });

// Query users collection ordered by age in descending order and limit to 2 documents
new FirestoreReference()
     .Collection("users")
     .OrderBy("age", OrderDirection.DESCENDING)
     .Limit(2)
     .RunQuery()
     .OnSuccess((documents) =>
     {
        Debug.Log($"Query fetched successfully : {JsonConvert.SerializeObject(documents)}
     })
     .OnError((error) =>
     {
         Debug.Log("Error: " + error.message);
     });
```

Compound queries

When performing compound queries, use the **Where** method with Field input rather than chaining multiple **Where** methods together.

```
// Query users collection where age is greater than 20 and isStudent is false
new FirestoreReference()
```

```
    .Collection("users")
    .Where(Filter.And(new List<Filter>()
    {
        Filter.Field("age", FieldOperator.GREATER_THAN, 20),
        Filter.Field("isStudent", FieldOperator.EQUAL, false)
    }))
    .RunQuery()
    .OnSuccess((documents) =>
    {
        Debug.Log($"Query fetched successfully : {JsonConvert.SerializeObject(documents)}
    })
    .OnError((error) =>
    {
        Debug.Log("Error: " + error.message);
    });
```