firebase

A quick reference for ABAP developers integrating with Firebase services, covering authentication, database, and cloud functions.

Firebase Setup and Authentication

Initial Setup

1. Create a Firebase Project:

- Go to the Firebase Console.
- Click 'Add project' and follow the steps to create a new project.

2. Obtain Firebase Configuration:

- In your Firebase project, go to 'Project settings'.
- Under 'Your apps', add a new web app (even if you're connecting from ABAP).
- Copy the Firebase configuration object, which includes `apiKey`, `authDomain`, `projectId`, etc.

3. Set up ABAP HTTP Client:

- Use `cl_http_client` to make HTTP requests to Firebase.
- Ensure your ABAP system can access the internet.

Authentication Flow

1. Get Authentication Token:

Use Firebase Authentication to sign in users (e.g., using email/password or Google Sign-In). Obtain the ID token after successful authentication.

2. Send Token to ABAP:

Securely transmit the Firebase ID token to your ABAP system (e.g., via a custom API).

3. Verify Token in ABAP:

Use the Firebase Admin SDK (if available for ABAP via a wrapper) or a custom JWT verification library to verify the token's authenticity and expiration.

Structure: Security Rules are defined in a JSON-like syntax and determine access permissions for your Firebase data.

Basic Syntax:

```
service cloud.firestore {
  match
/databases/{database}/documents/{documen
t=**} {
    allow read, write: if <condition>;
  }
}
```

Common Conditions:

- `request.auth != null`: Requires authentication.
- `request.auth.uid == userId`: Checks if the user ID matches.
- `request.time < timestamp.date(2024, 1, 1)`: Checks the current time.

ABAP Code Example (Simplified)

```
" Sample ABAP code to call Firebase
Authentication endpoint
DATA: lo_http_client TYPE REF TO
cl_http_client,
      lv_url          TYPE string,
      lv_response     TYPE string.

" Replace with your Firebase project's
API key
lv_url =
'https://identitytoolkit.googleapis.com/
v1/accounts:signInWithPassword?
key=YOUR_API_KEY'.

TRY.
    cl_http_client=>create_by_url(
EXPORTING url = lv_url

IMPORTING client = lo_http_client ).

    lo_http_client->request->set_method(
'POST' ).
    lo_http_client->request-
>set_header_field( name = 'Content-Type'
value = 'application/json' ).

    " Replace with user credentials
    DATA(lv_body) = |
{{"email":"test@example.com","password":
"password","returnSecureToken":true}}|.
    lo_http_client->request->set_text(
lv_body ).

    lo_http_client->send( ).
    lo_http_client->receive( ).

    lv_response = lo_http_client-
>response->get_text( ).
    " Process the response (e.g.,
extract ID token)
    WRITE:/ lv_response.

  CATCH cx_http_client_error INTO
DATA(lo_exception).
    WRITE:/ lo_exception->get_text( ).
ENDTRY.
```

Firebase Realtime Database Integration

Database Structure

Firebase Realtime Database stores data as a JSON tree. Design your data structure carefully, considering:

- **Data Relationships:** How data is related (e.g., parent-child).
- **Data Duplication:** Whether to duplicate data for faster retrieval.
- **Security Rules:** How to secure access to different parts of your database.

Reading Data

1. Construct the URL:	The URL should point to the specific node you want to read from the database. Include the `.json` extension to retrieve data in JSON format.
2. Make an HTTP GET Request:	Use `cl_http_client` to perform a GET request to the constructed URL. Include any necessary authentication headers (e.g., the ID token).
3. Parse the JSON Response:	Parse the JSON response from Firebase to extract the data you need. You can use ABAP's JSON parsing capabilities (e.g., `cl_sxml_string_parser`).

Writing Data

1. Construct the URL:	The URL should point to the node where you want to write data. Use `PUT` to overwrite the entire node and `PATCH` to update specific fields.
2. Create the JSON Payload:	Create a JSON string representing the data you want to write to the database.
3. Make an HTTP PUT/PATCH Request:	Use `cl_http_client` to perform a PUT or PATCH request to the constructed URL. Include the JSON payload in the request body and the authentication headers.

ABAP Database Example (Simplified)

```
" Sample ABAP code to write to Firebase Realtime Database
DATA: lo_http_client TYPE REF TO cl_http_client,
      lv_url          TYPE string,
      lv_response     TYPE string,
      lv_json_data    TYPE string.

" Replace with your Firebase project's database URL and node path
lv_url = 'https://YOUR_PROJECT_ID.firebaseio.com/users/user123.json'.

TRY.
    cl_http_client=>create_by_url( EXPORTING url = lv_url

IMPORTING client = lo_http_client ).

    lo_http_client->request->set_method( 'PUT' ).  " Use PUT to overwrite, PATCH to update
    lo_http_client->request->set_header_field( name = 'Content-Type' value = 'application/json' ).

    " Sample JSON data
    lv_json_data = |{{"name":"John Doe", "email":"john.doe@example.com"}}|.
    lo_http_client->request->set_text( lv_json_data ).

    lo_http_client->send( ).
    lo_http_client->receive( ).

    lv_response = lo_http_client->response->get_text( ).
    WRITE:/ lv_response.

  CATCH cx_http_client_error INTO DATA(lo_exception).
    WRITE:/ lo_exception->get_text( ).
ENDTRY.
```

Firebase Cloud Functions Integration

Calling Cloud Functions

1. Deploy Cloud Function:	Create and deploy a Cloud Function in your Firebase project using Node.js or Python.
2. Construct the Function URL:	The URL is typically in the format `https://<region>-<project-id>.cloudfunctions.net/<function-name>`.
3. Make an HTTP Request:	Use `cl_http_client` to make an HTTP POST request to the function URL. Include any necessary data as a JSON payload in the request body.
4. Handle the Response:	Process the response from the Cloud Function, which may be in JSON format. Handle any errors that may occur.

Passing Data to Cloud Functions

When calling a Cloud Function from ABAP, you can pass data as a JSON payload in the request body. Ensure the Cloud Function is designed to receive and process this data.

Example JSON Payload:

```
{
  "param1": "value1",
  "param2": 123,
  "param3": {
    "nestedParam": "nestedValue"
  }
}
```

ABAP Cloud Function Example

```
" Sample ABAP code to call a Firebase
Cloud Function
DATA: lo_http_client TYPE REF TO
cl_http_client,
      lv_url          TYPE string,
      lv_response     TYPE string,
      lv_json_data    TYPE string.

" Replace with your Cloud Function URL
lv_url = 'https://YOUR_REGION-
YOUR_PROJECT_ID.cloudfunctions.net/YOUR_
FUNCTION_NAME'.

TRY.
    cl_http_client=>create_by_url(
EXPORTING url = lv_url

IMPORTING client = lo_http_client ).

    lo_http_client->request->set_method(
'POST' ).
    lo_http_client->request-
>set_header_field( name = 'Content-Type'
value = 'application/json' ).

    " Sample JSON data to send to the
function
    lv_json_data = |{{"message":"Hello
from ABAP!"}}|.
    lo_http_client->request->set_text(
lv_json_data ).

    lo_http_client->send( ).
    lo_http_client->receive( ).

    lv_response = lo_http_client-
>response->get_text( ).
    WRITE:/ lv_response.

  CATCH cx_http_client_error INTO
DATA(lo_exception).
    WRITE:/ lo_exception->get_text( ).
ENDTRY.
```

Security Considerations

Token Management

Store Firebase ID tokens securely in ABAP. Consider using secure storage mechanisms to prevent unauthorized access.

- **Token Expiration:** Always verify the token's expiration time before using it.
- **Token Revocation:** Implement a mechanism to revoke tokens if necessary (e.g., when a user logs out).

Data Validation

Validate all data received from Firebase to prevent security vulnerabilities (e.g., injection attacks).

- **Input Sanitization:** Sanitize all data before using it in ABAP code.
- **Data Type Validation:** Ensure that data is of the expected type.

Firebase Security Rules

Use Firebase Security Rules to control access to your Firebase Realtime Database and Cloud Storage. Define rules based on user authentication and data content.

Example Security Rule:

```
{
  "rules": {
    "users": {
      "$uid": {
        ".read": "auth != null &&
auth.uid == $uid",
        ".write": "auth != null &&
auth.uid == $uid"
      }
    }
  }
}
```