Directus Proxy Documentation (Firebase ↔ Directus)

This document explains how the provided Node.js proxy mediates access to a Directus instance using Firebase authentication, a static Directus service token, and a set of path-based rules.

Overview

- Acts as a reverse proxy in front of Directus.
- Decides per-request whether to pass through, require Firebase auth, or allow public access.
- Injects a Directus static token for authorized/allowlisted requests so Directus enforces permissions.
- Optionally enforces per-user filtering on complaint lists.

Key Components & Variables

- DIRECTUS_URL – Base URL of your Directus server.
- DIRECTUS_STATIC_TOKEN – Long-lived Directus token used by the proxy when calling Directus.
- Firebase Admin SDK – Verifies incoming Firebase ID tokens.
- http-proxy – Forwards requests to Directus after the proxy's checks.

Path Lists

Restricted Paths

Require a verified Firebase ID token unless a Directus session cookie already exists:

```
/items/Complaint

/items/Complaint_main_category

/items/Complaint_ratings

/items/Complaint_sub_category

/items/Status_category

/items/Status_subcategory

/items/aboutUs
```

```
/items/device_tokens

/items/location

/items/notification

/items/users
```

Public Paths (Allowlisted)

Publicly readable without Firebase, but the proxy still attaches the Directus static token:

```
/items/terms_and_conditions

/items/ComplaintTimeline

/items/District

/assets

/files
```

Note: Directus serves file/video/image bytes from /assets/:id. Keep /assets in the public list for smooth media playback. /files is generally not used for bytes in Directus v9/10 but keeping it here is harmless if your setup references it.

Decision Flow

1. CORS preflight (OPTIONS) is answered immediately with permissive headers.
2. Fast-path passthrough: if there is no Authorization header OR a Bearer token fails Firebase verification, forward the request unchanged to Directus. This preserves Directus public endpoints and session-based auth.
3. If the URL matches a Public Path, mark it as public and force Authorization to the Directus static token, then forward.
4. If the URL matches a Restricted Path and there is NO directus_session_token cookie: verify the Firebase token, read the email, map to a Directus user, inject headers, replace Authorization with the static token, and optionally enforce per-user filtering on complaint lists.
5. All other requests are forwarded unchanged.

Header Injection to Directus

When a request is public-allowlisted or Firebase-verified, the proxy injects:

- Authorization: Bearer <DIRECTUS_STATIC_TOKEN>
- x-firebase-uid: <Firebase UID>
- x-firebase-email: <Email from token>
- x-directus-user-id: <Directus user id resolved by email>

Firebase → Directus User Mapping

6. Decode and verify Firebase ID token via Firebase Admin SDK.
7. Extract the email from the decoded token.
8. Query Directus: GET /items/users?filter[email]=<email>&limit=1 using the static token.
9. If a row is found, use its id as the Directus user id; otherwise return 403.

Complaint Ownership Enforcement

For GET /items/Complaint (list reads), if the request does not already specify filter[user], the proxy appends filter[user]=<directusUserId>. This ensures users only see their own complaints by default.

```
// Example transformation (conceptual)

/items/Complaint            → /items/Complaint?filter[user]=123

/items/Complaint?status=1   → /items/Complaint?status=1&filter[user]=123
```

CORS Behavior

- OPTIONS → 204 with Access-Control-Allow-* headers.
- Access-Control-Allow-Origin mirrors request Origin (or * if absent).
- Supports credentials when browsers send cookies.

Error Handling

- 401 – No/invalid Firebase token where required, or mapping failed.
- 403 – Email missing in token or no matching Directus user found.
- 502 – Upstream proxy error (Directus unreachable, etc.).

Files, Images & Video Playback

- Binary files (images/videos) are served from /assets/:id.
- Keep /assets in Public Paths so clients can fetch media without Firebase.
- Proxy still sends the static token—useful for private assets in Directus.
- For smooth video seeking, ensure the proxy forwards the Range header.

```
// Optional: forward Range for streaming

proxy.on('proxyReq', (proxyReq, req) => {

  const range = req.headers['range'];
```

```
  if (range) proxyReq.setHeader('range', range);
});
```

Extending the Rules

Make an endpoint public (no Firebase): add its prefix to publicPaths.

Protect an endpoint: add its prefix to restrictedPaths.

Exclude a single route from token injection (e.g., /users/me): handle it before the lists and do not mark the request as __publicAllowed or __firebaseVerified.

Examples

Public file via /assets

```
curl -i http://<proxy-host>:3000/assets/<file-id>
# → No Firebase required; proxy authenticates to Directus with static token.
```

Restricted read without Directus session

```
curl -H "Authorization: Bearer <FIREBASE_ID_TOKEN>"   http://<proxy-
host>:3000/items/Complaint
# → Proxy verifies Firebase, maps user, injects static token,
#    and appends filter[user]=<id> if missing.
```

Existing Directus browser session

```
curl -H "Cookie: directus_session_token=..."   http://<proxy-host>:3000/items/Complaint
# → Forwarded unchanged (session takes precedence).
```

Security & Deployment Notes

- Store DIRECTUS_STATIC_TOKEN and service account credentials in environment variables or a secret manager.
- Restrict network access so only your app environments can reach the proxy.
- Enable basic health checks and minimal logging for decisions (public/restricted/passthrough).
- Use PM2/systemd with proper restart policies.

Appendix: Current proxy.js (reference)

```
const http = require('http');

const httpProxy = require('http-proxy');

const admin = require('firebase-admin');

const { URL } = require('url');


// If you're on Node <18, install node-fetch and uncomment:

// const fetch = require('node-fetch');


const serviceAccount = require('./service-account.json');


//  Use env vars in production

const DIRECTUS_STATIC_TOKEN = 'Uzw3ZQDmA6COgqCnJOrjXgqq2X_D0ryL';

const DIRECTUS_URL = 'http://127.0.0.1:8055';


admin.initializeApp({

  credential: admin.credential.cert(serviceAccount),

});


const proxy = httpProxy.createProxyServer({

  target: DIRECTUS_URL,

  changeOrigin: true,

});


// Inject static token for (a) verified restricted requests OR (b) whitelisted public
ones

proxy.on('proxyReq', (proxyReq, req) => {

  if (req.__firebaseVerified === true || req.__publicAllowed === true) {

    proxyReq.setHeader('Authorization', `Bearer ${DIRECTUS_STATIC_TOKEN}`);
```

```
    if (req.__firebaseUID) proxyReq.setHeader('x-firebase-uid', req.__firebaseUID);

    if (req.__firebaseEmail) proxyReq.setHeader('x-firebase-email',
req.__firebaseEmail);

    if (req.__directusUserId) proxyReq.setHeader('x-directus-user-id',
String(req.__directusUserId));

  }

});



// Paths that require Firebase auth when no Directus session exists

const restrictedPaths = [

"/items/Complaint",

"/items/Complaint_main_category",

"/items/Complaint_ratings",

"/items/Complaint_sub_category",

"/items/Status_category",

"/items/Status_subcategory",

"/items/aboutUs",

"/items/device_tokens",

"/items/location",

"/items/notification",

"/items/terms_and_conditions",

"/items/users"

];




// Paths allowed publicly (no Firebase) but still send static Directus token

const publicPaths = ['/items/ComplaintTimeline', '/items/District', "/files",
"/assets"];



/** Directus user lookup by email */
```

```
async function getDirectusUserIdByEmail(email) {
  const url =
`${DIRECTUS_URL}/items/users?filter[email]=${encodeURIComponent(email)}&limit=1`;
  const resp = await fetch(url, {
    headers: {
      Authorization: `Bearer ${DIRECTUS_STATIC_TOKEN}`,
      'Content-Type': 'application/json',
    },
  });
  if (!resp.ok) {
    const text = await resp.text().catch(() => '');
    throw new Error(`Directus user lookup failed (${resp.status}): ${text}`);
  }
  const json = await resp.json();
  const row = Array.isArray(json?.data) ? json.data[0] : null;
  return row?.id ?? null;
}

/** Force filter[user]=<id> for list reads on /items/Complaint */
function enforceComplaintUserFilter(originalUrl, userId) {
  const u = new URL(originalUrl, 'http://placeholder');
  const parts = u.pathname.split('/').filter(Boolean);
  if (parts[0] === 'items' && parts[1] === 'Complaint' && parts.length === 2) {
    const hasExisting = [...u.searchParams.keys()].some(
      (k) => k === 'filter[user]' || k.startsWith('filter[user]')
    );
    if (!hasExisting) u.searchParams.append('filter[user]', String(userId));
  }
  return u.pathname + (u.search ? u.search : '');
}
```

```
const server = http.createServer(async (req, res) => {

  // CORS preflight

  if (req.method === 'OPTIONS') {

    res.writeHead(204, {

      'Access-Control-Allow-Origin': req.headers.origin || '*',

      'Access-Control-Allow-Methods': 'GET,POST,PUT,PATCH,DELETE,OPTIONS',

      'Access-Control-Allow-Headers': req.headers['access-control-request-headers'] ||
'content-type,authorization',

      'Access-Control-Allow-Credentials': 'true',

    });

    return res.end();

  }



  // ⚡ FAST-PATH: if NO token or token CAN'T be verified → bypass ALL proxy logic and
forward as-is

  const passthrough = () =>

    proxy.web(req, res, {}, (err) => {

      console.error('Proxy error:', err?.message);

      res.writeHead(502, { 'Content-Type': 'application/json' });

      res.end(JSON.stringify({ error: 'Bad gateway' }));

    });



  // Peek at Authorization header (but don't modify it)

  const authHeaderRaw = req.headers.authorization || '';

  const bearer = authHeaderRaw.replace(/^Bearer\\s+/i, '').trim();



  if (!bearer) {

    // No token at all → passthrough unchanged

    return passthrough();
```

```
  }

  try {
    // Try to verify. If this throws, we bypass.
    await admin.auth().verifyIdToken(bearer);
    // If verification succeeds, continue to your existing logic below.
  } catch {
    // Token present but NOT verified → passthrough unchanged
    return passthrough();
  }

  try {
    const url = req.url || '';
    const hasDirectusCookie = (req.headers.cookie ||
'').includes('directus_session_token');

    // Allowlisted public endpoints: no Firebase, still send static token
    const isPublic = publicPaths.some((p) => url.startsWith(p));
    if (isPublic) {
      req.__publicAllowed = true;
      // Ensure Authorization header is the static token
      req.headers.authorization = `Bearer ${DIRECTUS_STATIC_TOKEN}`;
      return proxy.web(req, res, {}, (err) => {
        console.error('Proxy error:', err?.message);
        res.writeHead(502, { 'Content-Type': 'application/json' });
        res.end(JSON.stringify({ error: 'Bad gateway' }));
      });
    }
```

```
    // Restricted endpoints (needs Firebase unless Directus session cookie exists)
    const isRestricted = restrictedPaths.some((p) => url.startsWith(p));
    if (isRestricted && !hasDirectusCookie) {
      const authHeader = req.headers.authorization || '';
      const firebaseToken = authHeader.replace(/^Bearer\\s+/i, '').trim();
      if (!firebaseToken) {
        res.writeHead(401, { 'Content-Type': 'application/json' });
        return res.end(JSON.stringify({ error: 'No token provided' }));
      }

      // Verify Firebase
      const decoded = await admin.auth().verifyIdToken(firebaseToken);
      const email = decoded.email;
      if (!email) {
        res.writeHead(403, { 'Content-Type': 'application/json' });
        return res.end(JSON.stringify({ error: 'No email in Firebase token' }));
      }

      // Map to Directus user
      const userId = await getDirectusUserIdByEmail(email);
      if (!userId) {
        res.writeHead(403, { 'Content-Type': 'application/json' });
        return res.end(JSON.stringify({ error: 'No Directus user found for this email'
}));
      }

      // Mark context for proxyReq hook
      req.__firebaseVerified = true;
      req.__firebaseUID = decoded.uid;
```

```
      req.__firebaseEmail = email;
      req.__directusUserId = userId;

      // Enforce ownership filter on list reads
      if (req.method === 'GET' && url.startsWith('/items/Complaint')) {
        req.url = enforceComplaintUserFilter(url, userId);
      }

      // Replace Authorization so Directus only sees the static token
      req.headers.authorization = `Bearer ${DIRECTUS_STATIC_TOKEN}`;
      req.headers['x-firebase-uid'] = decoded.uid;
      req.headers['x-firebase-email'] = email;
      req.headers['x-directus-user-id'] = String(userId);
    }

    proxy.web(req, res, {}, (err) => {
      console.error('Proxy error:', err?.message);
      res.writeHead(502, { 'Content-Type': 'application/json' });
      res.end(JSON.stringify({ error: 'Bad gateway' }));
    });
  } catch (err) {
    console.error('Auth/Filter failed:', err?.message);
    res.writeHead(401, { 'Content-Type': 'application/json' });
    res.end(JSON.stringify({ error: 'Invalid token or user mapping failed' }));
  }
});

server.listen(3000, () => {
  console.log('Proxy server running on port 3000');
```

```
});
```

```
Middleware on Direct us server:
```

Proxy Authentication Middleware for Directus

Overvie

We introduced a new Node.js proxy middleware running on port **3000** that sits in front of the Directus backend (running on port 8055). This proxy:

- Forwards requests as-is to Directus without modifying them.

- Validates Firebase authentication tokens only for a specified set of protected endpoints.

- Skips authentication if the `directus_session_token` cookie is present.

- Supports parallel requests efficiently.

- Handles JSON and non-JSON request bodies correctly.

Files Added

- `proxy.js`: The main proxy server implementing the auth logic.

- `service-account.json`: Firebase service account credentials file for token verification.

- `ecosystem.config.js`: PM2 ecosystem config to run both the proxy (port 3000) and Directus (port 8055).

Dependencies Added

- `express`

- `http-proxy-middleware`

- `firebase-admin`

- `cookie-parser`

- `body-parser`

Install via:

npm install express http-proxy-middleware firebase-admin cookie-parser body-parser

How to Add New Protected Endpoints

- Open `proxy.js`.

- Modify the `protectedEndpoints` array to include any new API paths that require Firebase authentication.

- Paths are matched with `startsWith`, so you can protect entire route groups (e.g. `/items/Complaint`).

Example:

```
const protectedEndpoints = [
  "/items/Complaint",
  "/items/Complaint_ratings",
  "/items/location"
];
```

Nginx Configuration

- Update your Nginx proxy to forward requests to the new proxy on port 3000 instead of directly to Directus on 8055.

Example snippet:

```nginx
location / {
    proxy_pass http://157.245.132.155:3000;
}
```