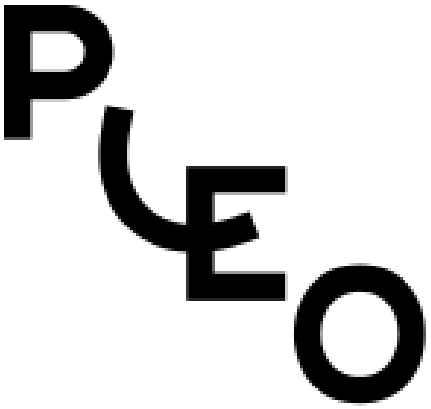

Pleo Technical Writing Task

Clarence Services API — Documentation & Improvement Proposal

Andrew Cleverly

October 2025

Preface – Submission for the Pleo Technical Writing Task

This document is my submission for the **Pleo Technical Writing Challenge**.

I have created a working, interactive documentation portal which will demonstrate Part 1 of the task to show how it should look, it includes:

- A complete **Introduction**, **Integration Guide**, and **OpenAPI schema**
- Functional navigation and mockable endpoints, viewable online at:
  Pleos Clarence API Portal
  I have ensure that this temporary URL is supported by HTTPS for security.
- The full file structure and authored source files are available on GitHub:
  Andrew Cleverly GitHub Repository

I chose this live, hosted format because **API documentation is a living system**, not a static deliverable. It evolves continuously through versioning, feedback, and integration testing — which a PDF cannot represent effectively. However, this PDF version is provided for review convenience for Part 1, I do request the link is reviewed as my main submission for Part 1 and then Part 2 is at the bottom part of this PDF.

This project also demonstrates the ability to **rapidly author documentation on a previously unused platform (Stoplight)**, adapting to its rendering rules and constraints to produce a structured, developer-ready experience.

The PDF was generated for the formatted Mark Down file using the Pandoc library for creation

Use of AI

AI (CoPilot) was used as an assistant to:

- Understand Stoplight's rendering syntax and platform-specific limitations (e.g., Mermaid/Markdown support)
- Accelerate layout prototyping and ensure consistent Markdown formatting

All writing, structure, and editorial tone were authored and refined manually.

Table of Contents

Part 1: Clarence Services API

Meet **Clarence** – Pleo's most popular (and occasionally napping) colleague.

This playful API lets you interact with Clarence through simple HTTP requests — no tennis balls required.

Use the API to:

- Tell Clarence to **sit**
- Ask him to offer a **paw** (choose left or right)
- Command him to **heel** and walk obediently beside you

When Clarence is asleep, your request will return a friendly error message so you can retry later — because even good boys need naps.

About this API

The Clarence Services API is a standard RESTful interface that shows how we approach developer-friendly design at Pleo: clear structure, meaningful responses, and documentation that's as approachable as the people behind it.

Every endpoint requires **OAuth 2.0** authentication with a Bearer token.
See Authentication for full details on obtaining and refreshing tokens.

What you'll find here

You can explore Clarence's API in two ways:

1. **Experienced with Pleo APIs?**
   Jump straight to the Quick Start and start calling endpoints in minutes.
2. **New to Pleo or APIs in general?**
   Head to the Documentation for a detailed, step-by-step walkthrough.

Sections overview

Section	Purpose
Documentation (Start Here)	**A guided walkthrough for new users**: From setup to first successful request, including authentication, example calls, and troubleshooting tips.
Authentication	**Detailed guide to OAuth 2.0**: Learn how to obtain, use, and refresh access tokens securely.
API Reference	**The full technical breakdown**: Endpoints for `sit`, `paw`, and `heel`, request and response structures, examples, and error definitions.

Why it matters

Clarence's API isn't just fun — it reflects our philosophy at Pleo:

Make complex things simple, make simple things delightful, and make sure the documentation feels like a friendly teammate guiding you through it.

Quick Start (3 minutes)

Get up and running with the Clarence API in a couple of simple steps.

> **New Users** Please follow the main guide if this your first time using this service HERE

> **Note:** This guide assumes basic familiarity with **OAuth 2.0** and access tokens.
> If you're new to OAuth, you can find a detailed walkthrough in the Authentication guide.

Option 1 : Try it in your browser (mock server)

You can experiment with Clarence's endpoints directly from this documentation.

1. Open the **API Documentation** page.
2. Select any endpoint under the **Clarence Actions** tag (`/clarence/sit`, `/clarence/paw`, or `/clarence/heel`).
3. Click **Send API Request** and choose the **Mock Server** environment.
4. Send a request to see realistic sample responses:
5. Change the response by selecting **Mock Settings** and then the response type.

Response	Meaning
200 OK	Clarence successfully performs the action (sit, paw, heel).
4 xx	The request is invalid or Clarence is asleep — you'll receive a friendly JSON error explaining why.

> The mock server is perfect for exploring request and response formats without needing real credentials.

Option 2 : Use it locally (Postman, Bruno, or similar)

Prefer to work in your own API-testing setup?

Download the complete OpenAPI schema and import it into your tool of choice.

1. Go to the **API Documentation page**.
2. In the top-right corner, click **Export** → **Bundled References**.
3. Your browser will download a single `.yaml` file containing the entire schema.
4. Import that file into **Postman**, **Bruno**, or any OpenAPI-compatible client.

Once imported, you can:

- Inspect all endpoints, parameters, and responses locally.
- Attach your own **OAuth 2.0 access token** in the `Authorization: Bearer <TOKEN>` header.
- Run live calls against the API or your mock server.

> Remember: production requests require a valid OAuth token.
> See Authentication for detailed steps on obtaining and refreshing tokens.

Authentication (OAuth 2.0)

Clarence's API uses **OAuth 2.0**. You'll call endpoints with an **access token** in the `Authorization` header.

Tokens from Pleo are **opaque** (do not decode).

- Access-token lifetime is returned as `expires_in`.
- Refresh tokens are valid **for at least 60 days**; using an **expired** refresh token invalidates all active refresh tokens for security.

Quick setup (Authorization Code flow)

Follow these steps to obtain your first access token and call an endpoint.

Prerequisites

Before starting the OAuth flow, make sure you've completed the following setup steps.

1. **Register your application**
   Create an OAuth client in the Pleo Developer Portal
   (or use this sample registration form) to obtain your:

   - `client_id`
   - `client_secret`
   - registered `redirect_uri`

   > These credentials identify your app when requesting tokens.

2. **Confirm redirect URI**

- Ensure your `redirect_uri` exactly matches what you'll use during the authorization flow (including protocol and path, e.g., `https://yourapp.com/callback`).

3. **Know your required scope**

   All Clarence endpoints require the scope:

```
clarence.actions
```

Request this scope when generating tokens.

Once these steps are complete, you're ready to begin the OAuth flow.

Authorization Code flow

1) Redirect the user to authorize

Send your user to Pleo's authorization page:

```
GET https://auth.pleo.io/oauth/authorize?response_type=code\
&client_id=YOUR_CLIENT_ID\
&redirect_uri=https://yourapp.com/callback\
&scope=clarence.actions\
&state=RANDOM_STRING
```

- Redirect_uri **MUST** exactly match a URL you control.
- State is any random string you'll verify later.

2) Handle the callback

After consent, Pleo redirects to your app with a short-lived code:

```
https://yourapp.com/callback?code=AUTH_CODE&state=RANDOM_STRING
```

Verify **state** matches what you sent.

3) Exchange the code for tokens

Swap the code for an access token (and refresh token):

```
curl -X POST "https://auth.pleo.io/oauth/token" \
  -H "Content-Type: application/x-www-form-urlencoded" \
  -d "grant_type=authorization_code" \
  -d "code=<AUTH_CODE>" \
  -d "redirect_uri=https://yourapp.com/callback" \
  -d "client_id=<CLIENT_ID>" \
  -d "client_secret=<CLIENT_SECRET>"
```

Example response

```
{
  "access_token": "3b4c78c0-12de-44c8-b5c3-1b9e21b38af9",
  "refresh_token": "0df6e8f9-8b4f-4b8a-93c7-27ccfab4fca7",
  "token_type": "Bearer",
  "expires_in": 3600,
  "scope": "clarence.actions"
}
```

Using your tokens

Once the `access_token` has been issued, include it in the `Authorization` header for every request to the Clarence API:

```
Authorization: Bearer <ACCESS_TOKEN>
```

- Access tokens expire after the duration specified in the **expires_in** field (typically 3600 seconds).

When this happens, use your **refresh_token** to request a new access token by sending a** refresh_token** grant.

When a refresh token is used successfully, a new refresh token is also issued.

You should replace the old one with the new value to stay in sync and prevent invalidation issues.

> **Always** Keep your tokens secure — never expose them in client-side code or version control.
>
> Use HTTPS for all token and API requests.
>
> Re-authorise from scratch if a refresh request fails, as expired refresh tokens invalidate all others for that app.

Integration Guide

This page is your guided setup:

1. Authenticate
2. Call an endpoint
3. Handle errors confidently.

The flow at a glance

The flow below shows the full process from authentication to calling an endpoint and handling responses.

The **first section** represents the Authenticate Process.

Follow that guide carefully to obtain your `access_token` before moving to the API calls.

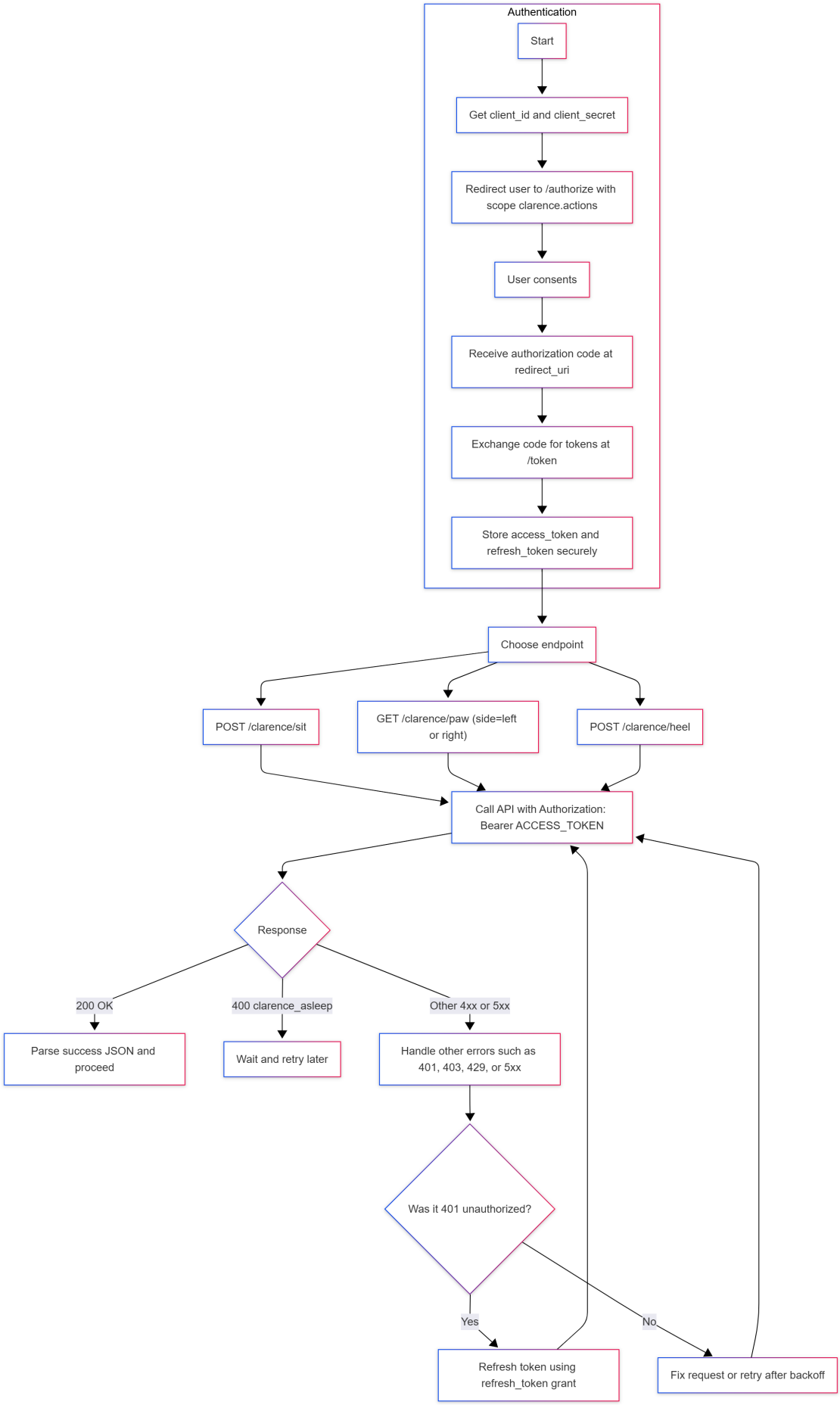

Figure 1: Flowchart

> **Important Essentials For Success** **Base URL:** `https://external.pleo.io/v3`
> **Version:** v3
> **Auth:** OAuth 2.0 → Bearer token (see **Authentication**)
> **Endpoints:** `/clarence/sit` (POST), `/clarence/paw` (GET), `/clarence/heel` (POST)

Step 1 — Authenticate

Before you can call any endpoint, you'll need an OAuth 2.0 access token.

Follow the Authenticate Process for a detailed walkthrough.
You'll:

1. Register your application and obtain a `client_id` and `client_secret`.
2. Complete the Authorization Code flow.
3. Exchange the code for an `access_token`.
4. Use that token in the `Authorization` header of every request:

```
Authorization: Bearer <ACCESS_TOKEN>
```

Step 2 — Call an Endpoint

Once authenticated, you can talk to Clarence through three endpoints.

POST /clarence/sit

> See API Documentation for full schema and example payloads.

Description Instructs Clarence to sit. This is an action command that changes his current state.

Request

```
curl -X POST "https://external.pleo.io/v3/clarence/sit" \
  -H "Authorization: Bearer <ACCESS_TOKEN>" \
  -H "Content-Type: application/json"
```

Successful Response (200 OK)

```
{
  "status": "ok",
  "action": "sit",
  "detail": "Clarence sits obediently."
}
```

GET /clarence/paw ?side=left | right

> □ See API Documentation for full schema and example payloads.

Description Asks Clarence to offer a paw. Requires a query parameter to specify which side.

Request

```
curl -X GET "https://external.pleo.io/v3/clarence/paw?side=left" \
  -H "Authorization: Bearer <ACCESS_TOKEN>" \
  -H "Content-Type: application/json"
```

Successful Response (200 OK)

```
{
  "status": "ok",
  "action": "paw",
  "side": "left",
  "detail": "Clarence offers his left paw."
}
```

If no side is provided, the default is right.

POST /clarence/heel

> □ See API Documentation for full schema and example payloads.

Description Tells Clarence to heel and walk beside you.

Request

```
curl -X POST "https://external.pleo.io/v3/clarence/heel" \
  -H "Authorization: Bearer <ACCESS_TOKEN>" \
  -H "Content-Type: application/json"
```

Successful Response (200 OK)

```
{
  "status": "ok",
  "action": "heel",
  "detail": "Clarence heels politely by your side."
}
```

Step 3 — Handle Errors

Even good dogs nap. When Clarence can't respond, you'll receive clear, actionable errors.

400 Bad Request – Clarence is asleep

Example Response

```
{
  "type": "clarence_asleep",
  "title": "Clarence is asleep",
  "status": 400,
  "detail": "Clarence is currently asleep. Try again in a few minutes or check
  ↪  his nap schedule."
}
```

Meaning Your request reached Clarence, but he's temporarily unavailable.

How to fix Wait a few minutes and retry the same request. No other action is needed.

Other Common Errors

HTTP Code	Error	What it means	How to fix
400 Bad Request	clarence_asleep	Clarence received your request but is currently asleep and cannot respond.	Wait a few minutes and retry the same request.
401 Unauthorized	invalid_token	The access token is missing, expired, or invalid.	Obtain or refresh the token, then retry.
403 Forbidden	insufficient_scope	The token does not include the required scope (clarence.actions).	Re-authorize with the correct scope.
404 Not Found	invalid_endpoint	The endpoint URL or resource does not exist.	Check you're using a valid path (e.g. /clarence/paw).

Part 2 – Content Improvement Proposal

Approach and Observations

If improving this documentation further, I would carry out the following steps:

- Use **analytics** and **developer feedback** to measure where users hesitate or drop off.
- While separating the Authentication flow makes sense for reuse, early analytics might show developers prefer a single, continuous "first-run" guide.
  If that were the case, I would merge the OAuth section directly into the Integration Guide so readers never lose context during their first setup.

To improve **ease of understanding**, I would:

- Review every paragraph for conversational clarity while keeping technical precision.
- Replace or remove redundant repetition across the Intro, Auth, and Integration pages.
- Re-validate every example personally to ensure the request/response data is current and correct.
- Reorganize long sections into collapsible topics for readability.

These steps align with Pleo's documentation values — approachable, human, and developer-first.

Brand & Tone

The current voice aligns with Pleo's friendly and transparent tone: plain English, positive framing, and clear next steps.
However, I'd ensure consistency in punctuation, headers, and use of emoji or humour depending on what part of information we are discussing.

API Naming & Usability

The Clarence API names are already clear (`sit`, `paw`, `heel`) — they convey action naturally.
If this were a production API, I'd confirm that:

- Verb-based naming conventions are consistent (e.g., POST actions vs. GET retrievals).
- Response objects share a common schema shape for predictability.
- Each endpoint includes clear success and failure modes (as already shown in the Integration Guide).

Minimising Support Requests

Each page should include:

- Working **curl** examples and copy-paste-ready snippets
- Meaningful, self-describing error messages with suggested fixes
- A "Try it" mock environment link for instant testing

This combination ensures developers can achieve their first successful call without opening a ticket, this can be seen by the mock up portal I used - linked in the preface.

Multi-Treat Configuration Feedback

The current text on **Multi-Treat Configuration** is informative but could be improved for clarity and tone.

Issues

- Phrases like "optimise treat exchange" are slightly abstract; developers may not understand operational implications.
- The FAQ mixes conceptual and procedural detail, which could be re-organised.

Improvements to FAQ Layout and Tone

Current Issues

- Inconsistent phrasing: "GET Get Treat Items" repeats words and feels unpolished.
- Dense blocks of text hide key facts; readers have to dig for meaning.
- Conceptual answers (how Multi-Treat works) and procedural answers (what to do) are mixed together.

Recommended Changes

1. Simplify and re-title the section to "Multi-Treat Configuration FAQ."
2. Group related questions under short sub-headings with bold lead-ins for scanability.
3. Correct naming to match REST standards and fix "GET Get " duplication.
4. Adopt Pleo's friendly, directive tone — straightforward, polite, and helpful.

Example Revision

Q1. What happens if Multi-Treats aren't enabled? If Multi-Treats are disabled, the `GET /v3/clarence/treat-items` endpoint returns a `400 Bad Request`:

```
{
  "type": "multi_treat_disabled",
  "title": "Multi-Treats not configured",
  "status": 400,
  "detail": "Please enable Multi-Treats in the Treat Settings page before calling
  ↪  this endpoint."
}
```

This mirrors the Clarence API error format and provides a clear next step.

Q2. What if an export job is already in progress when this error occurs?

You can configure your integration to either:

- Fail the corresponding export job immediately, or
- Wait until the job expires (after one hour since its last update).

Presenting the information in this structured way makes the FAQ faster to scan and easier to act upon.

Minimising Support Requests

To reduce developer frustration and tickets:

- Ensure every endpoint has a working example request and example response.
- Include explicit error explanations with practical fixes.
- Offer mock or sandbox environments for safe first-run testing.

This combination empowers developers to achieve their first successful call independently — a key measure of great API documentation.