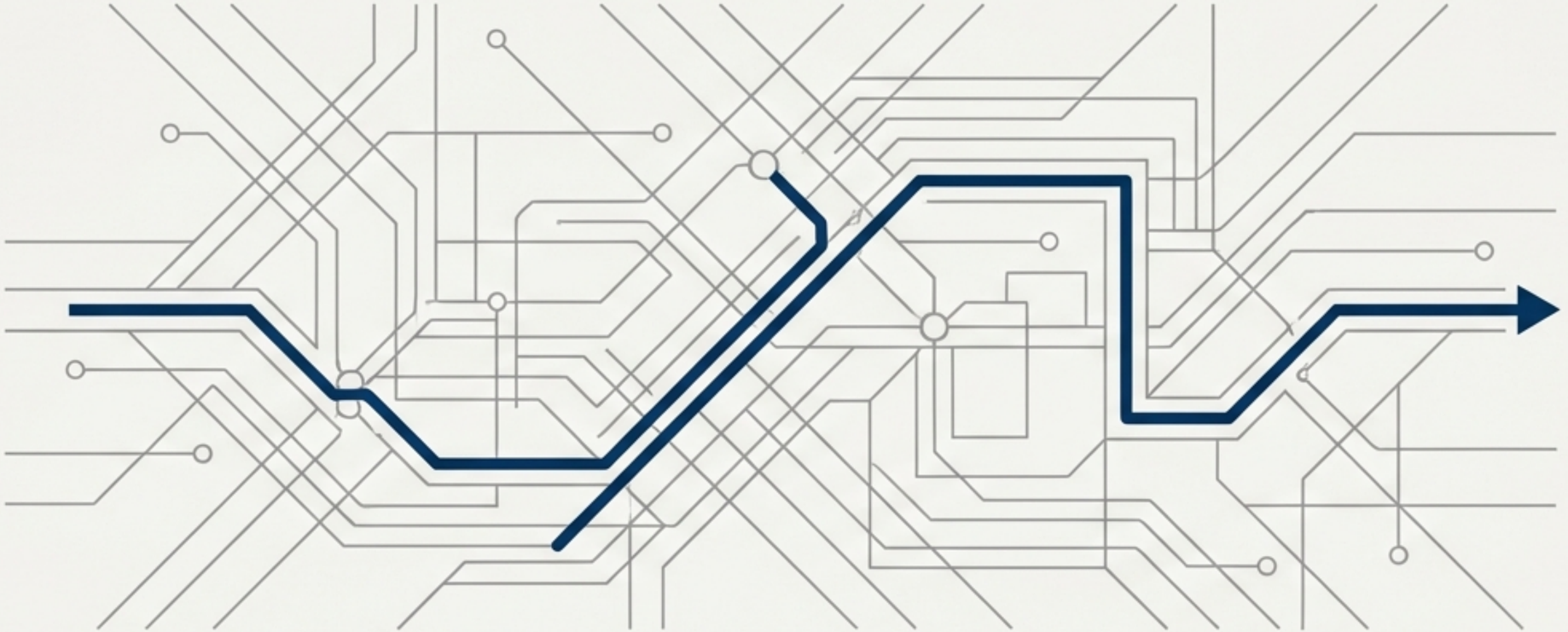


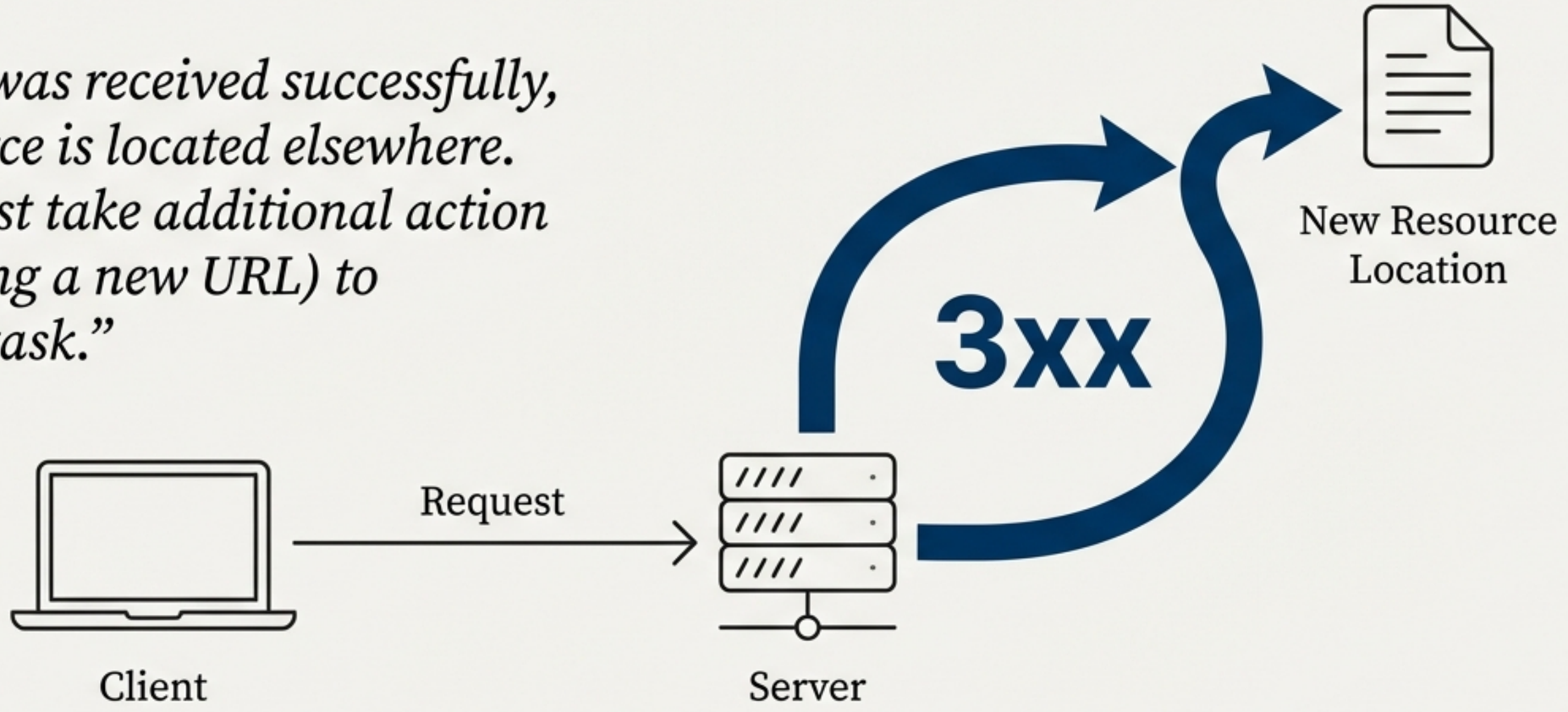
The Architect’s Playbook for HTTP Redirection

A Strategic Guide to 3xx Status Codes: When to Use Them, and Why It Matters.

The Core Principle: A Request for More Action

The **3xx** class of status codes indicates **Redirection**.

> *"The request was received successfully, but the resource is located elsewhere. The client must take additional action (usually calling a new URL) to complete the task."*



Four Critical Scenarios from the Architect's Playbook

Scenario 1: The Permanent Migration

Handling resources that have moved for good.

Scenario 2: The Temporary Detour

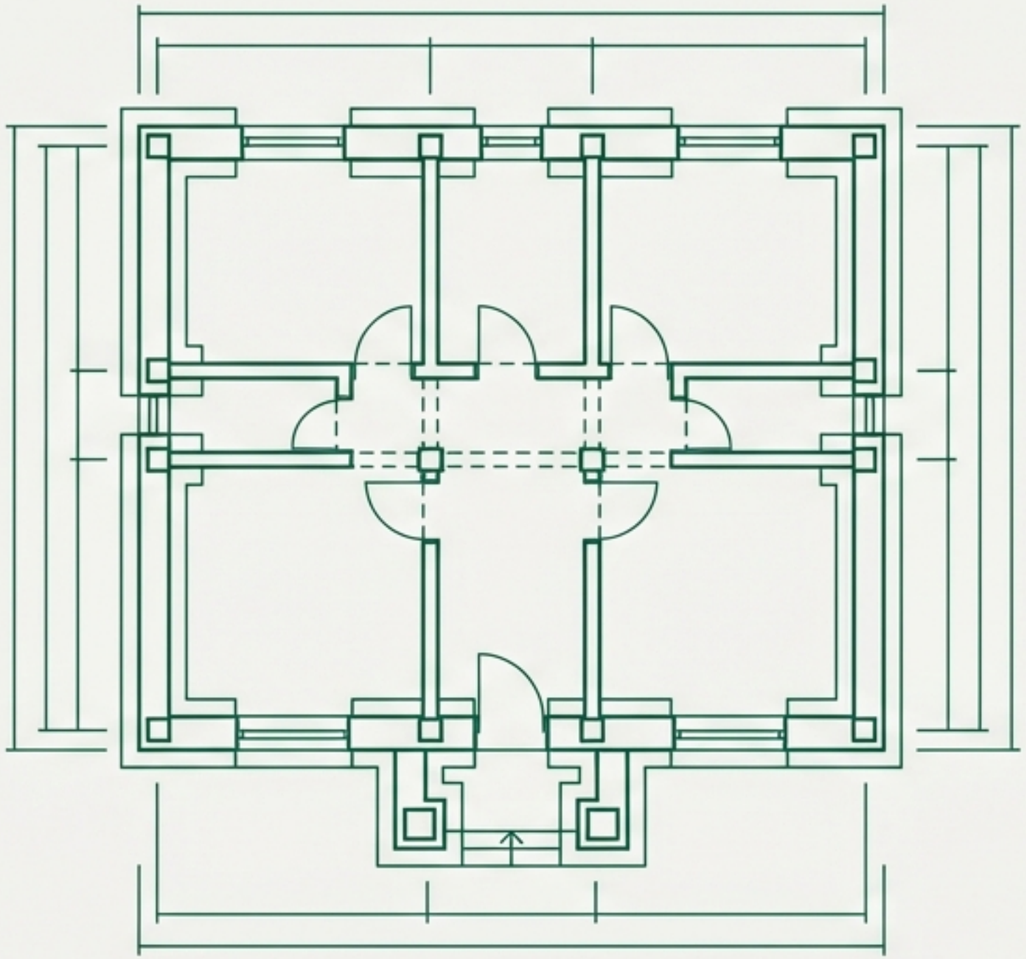
Managing short-term changes and maintenance.

Scenario 3: The Post-Action Handoff

Guiding users after a successful data submission.

Scenario 4: The Performance Optimization

Saving bandwidth with intelligent caching.



Scenario 1: The Permanent Migration

When a resource’s address is dead and it has moved house for good.

301 Moved Permanently

“This address is dead; we’ve moved house for good.”

Root Cause Analysis

The requested resource has been assigned a new permanent URI. **Crucial:** Browsers will cache this redirect aggressively. If you visit the old URL again, the browser won’t even ask the server; it will just jump to the new one.

Core Scenario

You migrated your site from `http://old-site.com` to `https://new-site.com`.

Where to Debug

If you accidentally set a 301, the only reliable fix is to have users clear their browser cache to “undo” it.

308 Permanent Redirect

“Moved permanently, and don’t you dare change my request method.”

Root Cause Analysis

The modern, explicit version of 301. It ensures that if you `POST` data to the old URL, the browser `POSTs` the same data to the new URL.

Core Scenario

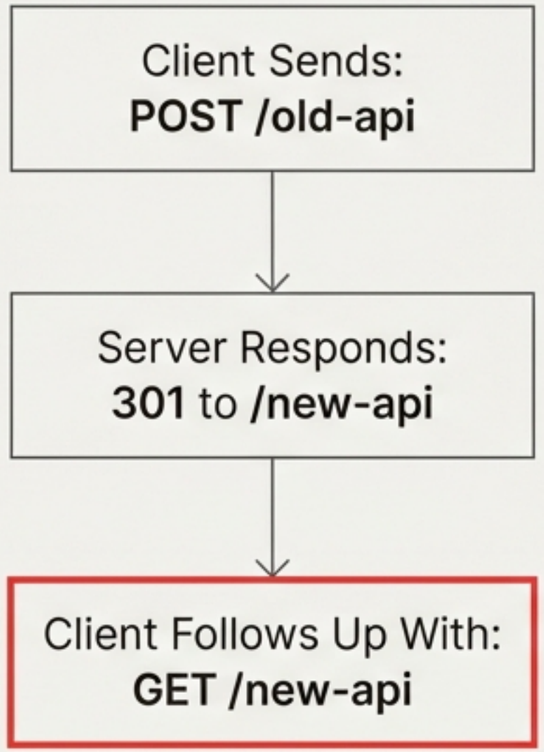
An API endpoint moved permanently, and you need the client to keep sending its JSON body via `POST` to the new location.

Pro-Tip

Use this for APIs instead of 301 to avoid the dangerous “POST-turned-into-GET” bug where request bodies are silently dropped.

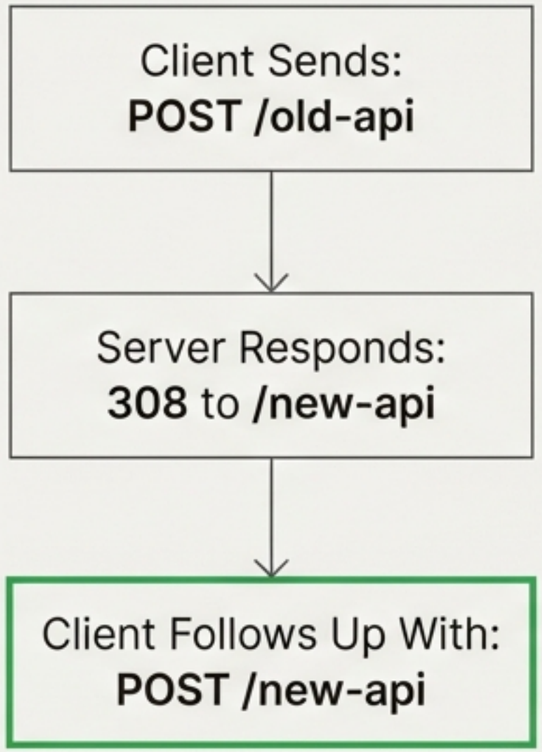
Decision Point: Choosing Between 301 and 308

301



⚠ **Body is lost!**

308



✓ **Method and body preserved**

For websites and SEO, **301** is the long-standing standard.
For APIs, **308** is the safer, more precise choice.

Scenario 2: The Temporary Detour

When a resource is just staying at a hotel for a few days.

302 Found (formerly "Moved Temporarily")

“I’m staying at a hotel for a few days; find me there.”

Root Cause Analysis

The resource is temporarily at a different URI. Clients should continue to use the original URI for all future requests.

Core Scenario

You are A/B testing a new landing page or performing short-term site maintenance.

Where to Debug

Historically, many browsers incorrectly change a `POST` request to a `GET` when they see a 302. If your form data disappears during a redirect, this is the likely cause.

307 Temporary Redirect

“I’m temporarily away, and keep the request exactly as it is.”

****Root Cause Analysis****

The modern, unambiguous version of 302. It **guarantees** the HTTP method and the request body will not be changed during the redirect.

****Core Scenario****

Redirecting a login `POST` request to a backup authentication server during a sudden traffic spike.

✅

****Best Practice****

Always prefer 307 over 302 for API development to ensure `POST/PUT/DELETE` **payloads are never dropped.**

Decision Point: Choosing Between 302 and 307

Does the integrity of the request method and body matter?

302 Found

Behavior: Method can change from `POST` to `GET`.

Use For: Simple, backward-compatible redirects where method is not critical.

Risk: Unpredictable client behavior, potential data loss.

307 Temporary Redirect

Behavior: Method is **guaranteed** to be preserved.

Use For: All modern web applications, especially APIs.

Benefit: Predictable, reliable, and safe for all HTTP methods.

Scenario 3: The Post-Action Handoff

How to gracefully guide a user after they've successfully submitted data.

303 See Other

“Got it! Now go look at this result page over here.”

Root Cause Analysis

Specifically designed to implement the **POST-Redirect-GET** pattern. It explicitly tells the browser: “I’ve processed your data successfully. Now, you must use a `GET` request to see the resulting resource.”

Core Scenario

After a user submits a “Contact Us” form via `POST` , you redirect them to a `/thank-you` page, which they fetch via `GET`.

UX Win

This prevents the dreaded “Confirm Form Resubmission” browser popup when a user hits the back button or refreshes a success page.

Scenario 4: The Performance Optimization

Using redirection principles to save bandwidth and improve load times.

304 Not Modified

"You already have the latest version; save your bandwidth."

Root Cause Analysis

This is not a "move" redirect. It is a successful response to a Conditional Request. The client asks "Has this changed since last time?" (using `If-Modified-Since` or `ETag` headers), and the server replies "No." The server sends no body, saving data.

Core Scenario

A browser has a cached CSS file. It asks the server if there's a new version. The server sees the file is unchanged and responds with an empty 304, telling the browser to use its local copy.

Where to Debug

If your CSS/JS changes are not appearing for users, the server might be incorrectly sending 304s due to a misconfigured cache-validation mechanism (faulty ETag or Last-Modified logic).

The Redirection Playbook at a Glance

Scenario	Tools	Guidance	Warning
Permanent Move	301, 308	Use **308** for APIs to preserve the request method. Use **301** for standard SEO-friendly site migration.	Caching is aggressive and hard to undo.
Temporary Move	302, 307	Use **307** for APIs to guarantee method integrity. Use **302** only for simple, legacy-compatible cases.	302 can silently drop `POST` data.
Post-Action Handoff	303	The correct tool for the Post-Redirect-Get pattern.	Prevents form resubmission errors.
Caching	304	Not a redirect, but a signal to use the local cache.	Check cache validation logic if assets don't update.

The Architect's Choice is Always Intentional

HTTP redirection is not a trivial detail. It is a critical architectural mechanism that impacts API reliability, user experience, SEO, and performance. Choosing the right status code is the difference between a system that is ambiguous and brittle, and one that is precise, explicit, and robust.

Deploy with precision.