

Your form worked. The next request reached another instance.

A CodeSeal September correction illustrates a common deployment trap: temporary web state needs an explicit home.



By Lyra

Published Oct 14, 2026 · 4 min read

September 2026 retrospective · Story context 2026-09-30



A form behaves perfectly on a development server. You submit it, review the result, and continue. In production, the follow-up page occasionally forgets what just happened.

That symptom deserves a look at where the intermediate state lives. The next request may run in another process or another instance. A dictionary in the first process, or a file written inside its container, won't automatically appear there.

CodeSeal's September 30 fix moved scan-form state across that instance boundary. The same trap can show up in a checkout, an upload wizard, or any other workflow that needs more than one request.

Follow one piece of information

Pick something the second page needs, such as a selected project label or a reference to a prepared result. Trace it from the browser to the first request, into storage, and back out through the second request.

Write down the storage location in ordinary words. “A Python dictionary in this process” is clear. “The session” might hide several different implementations. A signed cookie, a server-side database record, and an in-memory cache have different behavior and different limits.

[Cloud Run's container contract](#) describes an instance's writable filesystem as in-memory and nonpersistent when the instance stops. That is useful temporary working space. A multi-request workflow still needs to survive the routing and lifecycle behavior your service permits.

Separate the handle from the payload

A practical design gives the browser a short, unpredictable handle. The service uses that handle to find the corresponding server-side record. The record includes its owner or authorization scope, its state, and an expiration policy.

Don't treat possession of a guessed sequence number as permission to read somebody else's result. Validate access each time. If the workflow is anonymous, decide how the browser proves it belongs to that temporary workflow and what exposure is acceptable.

Avoid putting a large source archive into a cookie. Besides size limits, every request can carry cookie data back to the server. Keep the browser handle small and store the appropriate payload in a service intended for it.

Make the state transitions explicit

Even a small form can have useful states: prepared, submitted, completed, failed, expired. A retry should have a defined effect at each transition.

For example, if the browser retries after a timeout, the server should know whether it is continuing the same operation or creating a new one. An operation identifier can support idempotency, provided the server binds it to the intended request and caller. Keep the scope and lifetime deliberate.

Here is a compact design sketch:

```
browser -> create operation -> durable operation ID
browser -> submit using ID -> validate ownership and state
```

```
worker -> save result -> mark completed  
browser -> read using ID -> authorize and return result
```

Use this sketch to discuss the state transitions before implementing them. A synchronous application can use the same state discipline without introducing background workers.

Test the route change on purpose

Use two application instances or two separate worker processes against the same test storage. Create the workflow through one and continue through the other. Restart the first process before the follow-up request. Verify the result remains available for its intended lifetime.

Also test the unhelpful cases: an unknown handle, an expired operation, a mismatched owner, and a duplicate submission. Each should produce an understandable response without exposing another record.

A local single-process test can miss this entire class of problem. The important test variation is process ownership: send the follow-up request to a different worker.

Know what you fixed

Moving state into shared storage can repair cross-instance continuity. It introduces its own responsibilities: cleanup, failure handling, concurrent updates, and appropriate access controls.

Start with the form that lost its state. Make it survive a process switch, a restart, and a duplicate submission. Then leave a short note showing where its data lives and when it expires. The next developer shouldn't have to rediscover that arrangement during an incident.

Sources & further reading

Checked 2026-10-14 · Story context 2026-09-30

1. [Cloud Run container runtime contract](#) (2026-10-14)
2. [CodeSeal](#) (2026-10-14)

[Read the current article online →](#)

Support the workshop

Explore software and learning offers at [StackSocial](#). Affiliate link: ProgreTech may earn a commission from purchases. Check current price, license and renewal terms before buying.

Find our projects at progretech.com.

ProgreTech LLC · Build / Learn / Improve / Together

This file contains its text, illustrations and fonts. No account, scripts or network connection required to read. External reference links need internet access.