The Website Will Remember You If You Remember It

A build note on public website memory, private reader nodes, local-first storage, and why read state is not just a bookmark.

2026.07.12 02:45 DENNIS HEDEGREEN BUILD OPEN-NOTE
HTTPS://HEDEGREENRESEARCH.COM/ARTICLES/THE-WEBSITE-WILL-REMEMBER-YOU-IF-YOU-REMEMBER-IT/

The first version of website memory was public. An article needed to remember where it came from. A correction needed to point back to the claim it corrected. A Signal room needed to know that it was not an article. A PDF needed to know that reading on paper is not the same condition as reading on a screen.

That was the argument in `The Website Needs Memory Now`. The site had become more than a list of pages, so the public surface needed relations. Articles, sources, rooms, versions, receipts, and return paths had to stop floating alone.

That still matters, but it is no longer the whole problem. The reader has started leaving state behind. Not identity. Not a profile. Not a feed. State.

The article can know whether it has been read. Local can remember a surface mode. The PDF button can follow a reader's preference for standard, black and white, text-only, local, or source-facing output. Signal can hold room state. Notebook receipts can become part of a path instead of just another published object. Those are small features if each one is viewed alone. Together, they reveal the next layer of the website.

The site does not only need public memory. It needs a way to respect private memory.

A Bookmark Is Not Enough

A bookmark remembers an address. That is useful, but it is external to the website. It says: this URL exists, and I may want to go back.

A read state is different. A read state says: this public object has already happened to this reader. That is why the word "bookmark" is almost right and still too weak. The thing being remembered is not only a page. It is a relation inside the website.

An article can be unread, read, returned to, printed, saved, connected to a Signal track, or carried into a Notebook path. A source can be opened because a reader reached it through one article and later found it again through another. A PDF preference can become part of the reading condition. A surface mode can be a private accessibility choice, not decoration.

Those relations should not become public analytics. They should not be published beside the article. They should not become a hidden server profile. But they are real.

The simplest private node was not a note. It was a read state.

Public Nodes And Private Nodes

The public node was the easier half. An article is a public node. A Signal track is a public node. A TID object is a public node. A version, a source, an Atlas anchor, a Notebook receipt, and a PDF profile can all be public nodes.

They can be named, linked, corrected, related, and published. That is what the V2 site has been learning to do.

Archive, Investigate, Analyse, Build, Expose, Open, Journal, Signal, TID, Camelot. The rooms are not only navigation labels. They are a public memory system. They tell the reader what kind of object they are standing in.

But the reader's relation to those objects is not the same kind of thing. The private node belongs on the reader side: read, saved, return later, use text-only PDF, prefer negative mode, resume this Signal room, connect this article to that Notebook receipt.

None of that has to be public in order to be useful. The public graph says what the website has published. The private graph says what the reader has done with it. The mistake would be to collapse those two graphs into one.

Local Was The First Boundary

Local started as a practical place for choices. That is still what it should feel like. A reader should not need a theory of private nodes to change a PDF preference or switch a surface mode.

But Local is also the first boundary of reader-owned memory. It says: this state is here, in the browser, for the reader's use.

That matters because the first bad answer to website memory is always the same: make an account. Create a login. Store the path on the server. Call it personalization. Let the website remember because the website has captured the reader.

That is not the default I want here. The better first contract is smaller: visible, inspectable, erasable, exportable. Visible means the reader can see that state exists. Inspectable means it is not a hidden profile. Erasable means it can be cleared without asking a server for permission. Exportable means the reader can take it with them.

That last word changes the category. If read state can only live in one browser, it is convenience. If it can be exported and imported, it becomes memory. Still private. Still reader-owned. Still not public. But no longer disposable.

This is not a new problem in software. Ink & Switch, Martin Kleppmann, Adam Wiggins, Peter van Hardenberg, and Mark McGranaghan gave it a clearer name in 2019: local-first software. Their formulation was broader than this website, but the contract is close to the one I need here. Data should live on the user's device first. The user should be able to keep it, move it, and use it without a company server becoming the owner of the path.

Hedegreen Research is not inventing that lineage. It is running into a small website version of it.

The Browser Can Forget

Local browser storage is a good beginning because it lets the site become useful without demanding identity. It is not a durable archive.

Safari makes that tension visible. In WebKit's March 24, 2020 post `Full Third-Party Cookie Blocking and More`, Intelligent Tracking Prevention is described as deleting a website's script-writable storage after seven days of Safari use without user interaction on the site. LocalStorage is named in that affected group, beside IndexedDB, media keys, SessionStorage, and service worker registrations and cache.

That is not an attack on the browser. It is the browser doing privacy work. But it creates a real design problem for local-first websites. If the reader does not come back, the browser may eventually stop treating that site's local state as something to preserve. The state can be cleared or limited by policy, context, and use.

The same WebKit post also makes the architectural point sharper: web applications added to the home screen are treated differently from ordinary Safari browsing, because their use counter follows actual use of the web application. That matters. A site that becomes more like a local tool gets a more durable memory condition than a site visited as a disposable page.

So the title is literal. The website will remember you if you remember it.

At the first layer, memory depends on return. That is acceptable for a beginning. It is not enough for the whole system. If private nodes matter, they need a way to leave the fragile browser box without leaving the reader's control.

Memory Does Not Have To Mean Tracking

The site does not need to know who a reader is in order to help the reader keep their own path. It needs a local memory layer that is honest about itself.

No hidden analytics event pretending to be a feature. No public graph polluted with private reading behavior. No account system as the price of remembering. No server-owned profile disguised as convenience.

The private node should remain private because it is the reader's relation to a public object. That relation can be simple: read. It can be practical: open this article in text-only PDF. It can be directional: return here later. It can be interpretive: this belongs with that Signal track. It can become personal: this is where the point landed.

The website should not absorb that into its public structure. The private layer should not leak it back as performance data. The right move is to make the boundary explicit. Public nodes are published. Private nodes are owned.

The POD Question

This is where PODs become interesting again. Not as a slogan. Not as a way to bolt a login onto the site. Not as a vague promise that everything will be portable someday.

The narrow question is better: where should private nodes live when the browser is too fragile and the website should not own them?

Solid points at one answer. A POD is a user-controlled online data store. Apps can read and write with permission, and the user's data does not have to be locked inside one platform.

That is the direction that matters here, with one honest caveat: Solid has been an important idea for years without becoming the default way people carry website state. That does not make it useless. It means this site should not wait for Solid adoption before it learns how to export its own private nodes.

Portability Is The Entrance, Not The Exit

The first step is more ordinary: export, import, a private node bundle. That bundle could hold read state, saved internal nodes, PDF preferences, surface choices, Signal positions, return points, and private relations. Later, that bundle can become POD-compatible or live in another user-owned storage layer.

But portability is not the finish line. It is the entrance to the sync problem.

If a reader exports on a laptop, reads three articles on a phone, then imports the old laptop bundle, the site has to know what kind of private node it is merging. Read state is forgiving. A union is usually enough. If either device says the article was read, it was read. But a return point is not always mergeable that way. A private relation may have been corrected, moved, or deleted. A note can fork. A preference can conflict.

That is why the private node schema matters. It is not only a storage format. It is a promise about what can be merged automatically, what needs a timestamp, what needs a device origin, and what should ask the reader before overwriting anything.

It also has to know that private nodes do not all carry the same risk. A PDF preference is low sensitivity. A read timestamp is more sensitive. A private note attached to a public source can be very sensitive. If the private layer ever becomes portable, the schema has to carry that gradient instead of pretending all local state is equally harmless.

The important thing is to design the private layer so it can leave. If the reader owns the memory, the memory must be portable before it becomes clever.

What Comes Next

The next build step is not a social feature. It is a private node schema. The site should be able to describe a reader-owned relation without publishing it and without sending it to a server by default.

Read state should become a first-class object. Saved internal nodes should be different from browser bookmarks. Local preferences should be exportable. Signal should be able to say: you were here.

A small node could be as plain as this:

```
{
  "type": "read",
  "target": "A0152",
  "when": "2026-07-12T10:04:00+02:00",
  "surface_mode": "negative",
  "pdf_profile": "text-only",
  "sensitivity": "private-activity",
  "origin": "local-browser"
}
```

That is not the whole schema. It is the first honest shape: a private relation, a public target, a time, a surface mode, an output preference, a sensitivity level, and an origin.

The article surface should be able to say: you read this, and this source is where you may want to return. Notebook receipts should know when they belong to the public graph and when they belong to the reader's private path. Local should be able to say: here is the memory this browser is holding, here is how to erase it, and here is how to take it with you.

That is not personalization as a business model. It is a website becoming honest about what reading already is. Reading is not only access to a document. Reading creates state.

The old web mostly left that state split apart. The browser had bookmarks. The server had logs. The reader had memory. None of them were the same object.

This website is starting to show a fourth possibility. Public nodes can stay public. Private nodes can stay private. The relation between them can be useful without becoming extractive.

A bookmark says a page exists.

A read state says the page happened to someone.

That is small. It is also enough to change what a website is.

RELATION MEMORY

MATURES FROM The Website Needs Memory Now · Moves the public relation-memory argument into reader-owned private nodes.

RETURNS TO This Stopped Being Just One Website · Returns to the room-family build argument from the reader-state side.

SYSTEM REQUIREMENT The Executive Function Prosthetic: A System That Must Learn to Forget · Both pieces separate useful memory from keeping everything forever.

CANONICAL URL: HTTPS://HEDEGREENRESEARCH.COM/ARTICLES/THE-WEBSITE-WILL-REMEMBER-YOU-IF-YOU-REMEMBER-IT/
PDF VIEW GENERATED: 2026-07-12T00:53:23.407Z