BWEB

Architecture, Security & Technical Choices

bspoke

4D Method User Group — 2026

ACT 1

Built-in Security

Automatic zero-trust, without writing a single line

The security pipeline

Every request goes through 5 steps before reaching business logic

HTTPS

Auto redirect if cert.pem

License

Component check + session

URL Resolution

Domain → Page → Params

Hook

Host database can intercept

Security Check

Rights, pages, events, POST

> *The developer writes nothing. The component handles the entire pipeline.*

3 levels of rights

DECLARATIVE — CONFIGURED IN THE DEV PANEL, NOT HARDCODED

🔒 Level 1 — Domain

WebDomain.userRightUuid — Protects the ENTIRE site

📄 Level 2 — Page

WebDomainMenu.userRightUuid — Protects ONE page

👓 Level 3 — Block / Field

blockProperties.userRightNeeded — Hides a block or rejects its POST data

Security Check — Cascade

GG_WEB_SOCKET_SECURITY_CHECK : THE CENTRAL GATEKEEPER

❌ Unknown domain
→ 400

🚫 Unpublished page
→ 403

🔀 Configured redirect
→ 301/302

🔒 Protected domain / page
Not logged in → login | No permission → 403

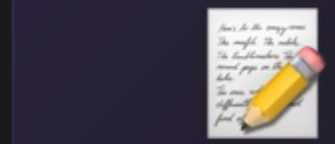
👾 Malformed URL (bot)
→ 404

✅ Everything OK
→ Business logic executed

Action validation

Every button click / form submission is verified

WebContent
Find the source block in the database



Event declared?
className + functionName

Constraints
Rights, required, regex, types

POST Filtering
Removes undeclared fields

> *Impossible to call an arbitrary function or inject fields.*

Automatic protections

Unauthorized access
✓ 3 levels of rights (domain / page / block)

Field injection
✓ Only fields declared in WebContent pass through

Arbitrary function call
✓ Server whitelist (className + functionName)

Man-in-the-middle
✓ HTTPS enforced if cert.pem present

Bots / scraping
✓ Malformed URLs → 404, form captchas

Protected block visible
✓ Block never sent to the browser (no CSS hide)

ACT 2

BWEB + AI vs Raw HTML

AI builds fast. BWEB makes it maintainable.

The workflow

Prompt

Describe what you need

Claude API

MCP BSPOKE + BWEB docs

BWEB Objects

wrapper, text, button...

Dev Panel

Editable by anyone

> *Not a dead HTML file. A living structure in the 4D database.*

The 6 advantages

Free security

5-step automatic pipeline. Zero code to write.

Human maintenance

Changing text = clicking in the Dev Panel. No code.

Native 4D data

Listbox → dataclass. Forms → ORDA entities.

MCP : best of both

AI generates HTML (its strength), MCP converts to BWEB.

CMS included

Domains, SEO, redirects, headers, publishing...

Secured events

Undeclared event in database = rejected. No arbitrary function.

Comparison table

	RAW HTML BY CLAUDE	BWEB + CLAUDE VIA MCP
Security	Code it yourself	✓ Automatic
Maintenance	Source code	✓ Visual Dev Panel
4D Data	Custom REST	✓ Native ORDA
CMS	From scratch	✓ Included
Rights	Code it yourself	✓ 3 declarative levels
Validation	Code it yourself	✓ Regex, types, required
Events	✕ Not secured	✓ Server whitelist
Publishing	Manual deploy	✓ Draft / published

Maintenance: human vs code

Raw HTML

Open the source code

Understand Tailwind classes

Edit, test, redeploy

Requires a developer

AI may rewrite the whole page instead of fixing one thing

Rollback = ask AI to rewrite again. Risky.

VS

BWEB

Open the Dev Panel

Click on the object

Change the value, it's live

Anyone can do it

AI changes one block, not the whole page

Undo / Redo built-in

ACT 3

BWEB vs React

Why we chose POST/WebSocket + server-rendered HTML

Two architectures

BWEB (Chosen option)

> 4D generates the full HTML
> Browser receives ready-made HTML
> POST + WebSocket for actions
> The server decides EVERYTHING

VS

React + REST (Discarded option)

> 4D exposes a REST API
> React builds the interface
> fetch() for every interaction
> The client decides, the server reacts

Our audience

4D DEVELOPERS, NOT JAVASCRIPT DEVELOPERS

React would require	BWEB requires
JSX / TSX	4D
useState, useEffect, hooks	Tailwind CSS
React Router	That's it.
Redux / Zustand	
npm, webpack, Vite	
TypeScript	

The server controls everything

React (client-side)

The client decides what to display

The client decides what to fetch

Protected blocks = hidden with CSS

Visible in the source code

VS

BWEB (server-side)

The server decides what to send

Blocks filtered by rights

Protected blocks = not in the HTML

Impossible to inspect

Not allowed to see a button? The button's HTML doesn't exist.

WebSocket vs React Callbacks

React after an action

Receive the API response

Update the state

Trigger a re-render

Handle errors

Invalidate cache (SWR...)

VS

BWEB after an action

```
reloadBlock("suppliersList")
sendAlert("success")
```

The server decides what to refresh

The client obeys. No state.

License and dependencies

💰 No REST license

POST/WebSocket goes through the standard 4D web server. For 300K users, the cost difference is enormous.

📦 Zero frontend dependencies

No React, Angular, Vue. No 200KB+ bundle. No breaking changes React 17 → 18 → 19.

🔄 Node.js = server tool

Tailwind, Puppeteer (PDF), scripts. Browser side: HTML + Tailwind + vanilla JS.

📈 Entity on the server

A single source of truth. No client copy. STAMP handled natively by ORDA.

Summary

	IF BWEB USED REACT	BWEB AS IT IS
Skills	4D + React + TypeScript	**✓ 4D only (Tailwind is a plus, not required)**
License	4D REST required	**✓ 4D Web is enough**
Security	Client decides	**✓ Server decides everything**
Protected blocks	**✕ Hidden with CSS**	**✓ Not in the HTML**
State	Client (Redux, hooks)	**✓ Server (process vars)**
Real-time	Code it yourself (Socket.io...)	**✓ Built-in WebSocket**
Dependencies	**✕ node_modules**	**✓ None**

The 3 pillars

Automatic security

AI as engine

Server controls everything

> *"We didn't avoid React out of ignorance. We avoided it by choice.*
> *A 4D developer should be able to create a website without changing worlds."*

The server generates, the server protects, the server controls.
The browser displays. That's it.