v0.63

Foundational Feature Server

Architecture, Components & Value Proposition

Technical Deep-Dive
Decision-Maker Edition

C++23 | Wt 4.12 | CMake | Container-Ready

Fehmi Demiralp
demiralp@fedem.eu

Agenda | What We Will Cover Today — 60 to 90 Minutes

Part I — Platform

01 Vision & Goals
02 Why C++?
03 Architecture Overview
04 FFS Core Layer
05 Component System
06 Data & Config Layer

Duration: ~25 min

Part II — Components

07 Component Map
08 Sliding
09 Deliverables & Services
10 Portfolio
11 Blog
12 Article
13 Contact
14 About & Testimonials
15 SDK Deep-Dive

Duration: ~35 min

Part III — Value

16 Theming System
17 Security Design
18 Container Toolchain
19 Build & Packaging
20 Multi-Language i18n
21 Competitive Matrix
22 5-Phase Roadmap
23 Value Proposition
24 Constraints & Q&A

Duration: ~25 min

Part I

Platform Overview

Vision · Architecture · Technical Foundation

Core Definition

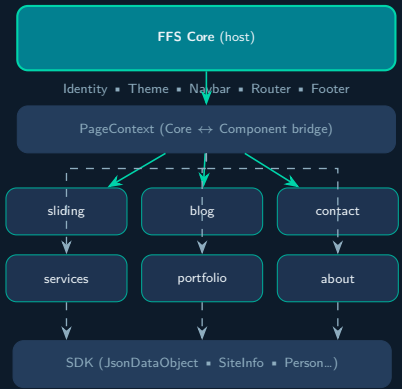
FFS is a **C++23 web-site creation and publishing platform** built on **Wt (Webtoolkit) 4.12**.

Pages and features are delivered as **independently developed components** — shared libraries (.page files) that can be added to or removed from a running site **without rebuilding the core**.

Design Philosophy — Five Pillars

- ✔ **Zero rebuild** to add or remove a page component
- ✔ **Zero downtime** content updates via live-reload
- ✔ **Zero JavaScript framework** — server-side C++ UI
- ✔ **four themes** (default / ocean / forest / mistery) built-in — extensible
- ✔ **Multi-language** EN / DE / TR — extensible

ARCHITECTURAL OVERVIEW



Performance

- Native binary — **no VM, no JIT**
- Sub-millisecond page generation
- Minimal memory per session
- Wt handles WebSocket + HTTP/2
- No GC pauses, no event loop lag

Safety & Control

- C++23 value semantics
- RAII resource management
- No `npm` supply-chain risk
- **Only 3 direct runtime deps**
- Compile-time error detection

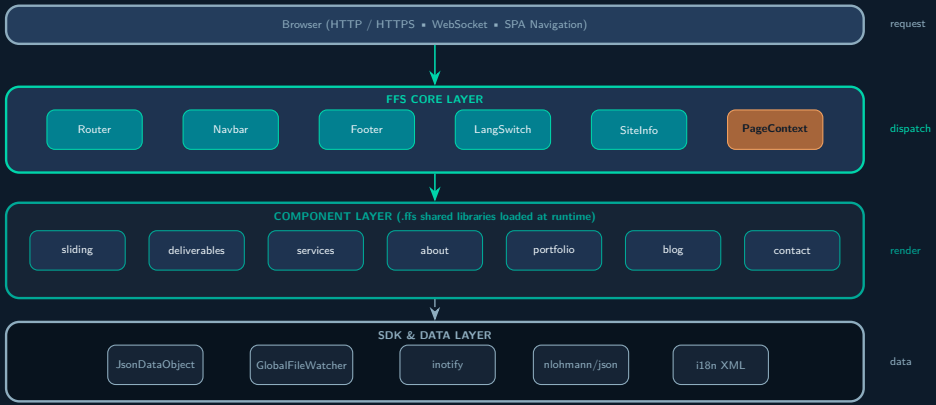
Comparison at a Glance

	FFS	Node	PHP	Hugo
RAM idle	**8 MB**	60 MB	30 MB	5 MB
Deps	**3**	500+	varies	~50
Build art.	**binary**	bundle	runtime	static
Type safety	**C++23**	TS opt.	none	none
Hot reload	**.so**	process	script	none

✔ **Key insight:** C++ here is *not* a complexity cost — it *reduces* runtime uncertainty. Build once, deploy anywhere with deterministic behaviour.

▲ *Approximate values — FFS figure to be measured; others based on published benchmarks and general knowledge.*

Architecture at a Glance | Four Layers, One Binary, N Independent Components



Core Modules

Router SPA-style URL dispatch. Maps `/{locale}/{page}` to the correct component. Handles 404 and maintenance mode gracefully.

Navbar Responsive navigation. Menu order and labels driven by `site.json` — **zero recompile** to reorder or hide any page.

Footer Auto-populated from ServicesData. Services with `in_footer:true` appear automatically.

LangSwitcher EN/DE/TR toggle. Cookie written *only* on explicit user click — no surprise overrides.

PageRegistry Discovers and loads `.ffs` shared libraries at startup via `site.json` page list.

PageContext Bridge: gives each component appRoot, locale, siteInfo, replacer, and session context.

Locale Resolution — Priority Chain

#	Source	Writes cookie?
1	URL prefix `/tr/about`	✘
2	`language` cookie	—
3	`Accept-Language` HTTP header	✘
4	`default_language` in `site.json`	✘
5	First entry in `languages[]` array	✘
6	Hardcoded fallback `"en"`	✘

site.json — page visibility

```
{ "default_language": "de",
  "languages": ["de","tr","en"],
  "pages": [
    {"name":"sliding",  "show":true,  "display":""},
    {"name":"blog",     "show":true,  "display":""},
    {"name":"contact",  "show":false, "display":"Contact"}
  ]
}
```

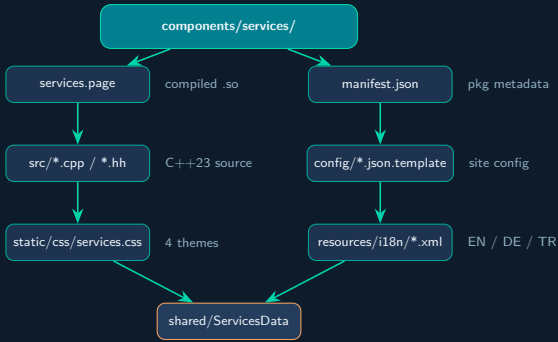
What is a .ffs Component?

Each component is a **self-contained unit** that ships:

- **.ffs** — compiled shared library (`.so`)
- **manifest.json** — declares dependencies and capabilities
- **config/** — JSON templates (site-specific overrides)
- **content/** — locale-overlay JSON + Markdown
- **static/css/** — per-theme CSS (`<name>.css`)
- **resources/i18n/** — XML translation bundles

Components are **loaded once at server start** and serve all sessions — **zero per-session cost**.

▲ **Component isolation guarantee:** A component cannot break the Core, cannot affect another component's state, and requires no platform restart to add or remove (Phase 2). Isolation is by architectural design.



Live Reload — GlobalFileWatcher

One inotify thread for the entire process. Change event → `WServer::post()` → correct session.
Production: all watchers are no-ops — zero threads, zero fds.

Data & Configuration Layer | JsonDataObject + Locale-Overlay System

JsonDataObject — Base Class

Foundation for all config and content objects.

Method	Purpose
`load(appRoot)`	Parse JSON + start inotify watch
`loadWithLocale(..)`	Base + locale overlay merge
`save(appRoot)`	Atomic write (.tmp then rename)
`deepMerge(dst,src)`	Nested + array-wise merge
`addNotifier(fn)`	Returns NotifierHandle (RAII)
`stopWatch()`	Cleanup — must be first in dtor

Correct notifier pattern (C++)

```cpp
// Capture WApplication* in constructor
auto* wtApp = Wt::WApplication::instance();
handle_ = slides_.addNotifier([this, wtApp] {
    Wt::WApplication::UpdateLock lock(wtApp);
    if (!lock) return;
    onDataChanged();
    wtApp->triggerUpdate();
});
// Destructor: handle_ goes out of scope -> auto cleanup
```

Locale-Overlay JSON System

```
services.json        <- base (EN fallback)
services_de.json     <- DE overlay (translatable only)
services_tr.json     <- TR overlay
```

Load sequence for a DE site:

1. `services.json` → `fromJson()` → base object
2. `services_de.json` → `deepMerge()` → DE values override
3. Keys absent from DE → fallback to EN **(automatic)**

Strict rule: Structural keys (`show`, `anchor`, `icon`, `in_footer`) live *only* in the base file. Locale files carry *only* translatable strings.
Clean separation of structure and content.

✔ **Zero coupling:** Adding a new language requires only a new `*_<lang>.json` file — no recompile, no restart.

Part II

Components Deep-Dive

Each Page ▪ Its Architecture ▪ Its Value

Component Ecosystem | 10 Independent .page Shared Libraries

Purpose & Features

The **first visual impression** of any FFS site.

- Full-width hero slideshow — **10 transition types**
- Per-slide **publish / expire** scheduling with timezone offset
- Per-slide duration override; global fallback `duration`
- **followme tiles** — social and link shortcuts below hero
- SVG icon with emoji fallback; dot navigation (`z-index:20`)
- URL support: external (new tab) or internal (SPA route)
- **Live-reload:** new image on disk → auto-discovered + cache-busted `?v={mtime}`

slides.json — structure

```
{ "duration":5, "corner_radius":12,
  "transition":{"type":"fade","speed":600},
  "pictures":[
    { "name":"slide-01",
      "path":"static/images/slides/slide-01.jpg",
      "show":true,
      "publish":"20260601 09:00 +0200",
      "expire":"20261231",
      "caption": {
        "position": "lower-left",
        "title": "Industrial Automation",
        "subtitle": "Robust Real-Time Systems"
      }
    }
  ]
}
```

Transition Types (10)

`fade`	`slide-left`	`slide-right`	`slide-up`	`slide-down`
`zoom-in`	`zoom-out`	`flip-h`	`flip-v`	`none`

- ✔ Empty slide list → idle polling `[2s–10s]`, timer never stops.
- ✔ Locale overlay: only `title`/`description` fields in `slides_de.json` — structural keys stay in base.

DeliverablesPage

Purpose: Quick visual inventory of service/product offerings — icon tiles in a configurable **N×M grid**.

- `column` × `row` grid layout (JSON-configured)
- Per-tile: icon, emoji fallback, label, target URL
- Click → SPA navigate to service anchor
- Live-reload via `addNotifier`

```
{ "column": 4, "row": 3,
  "card_min_width": 140, "card_max_width": 200,
  "deliverables":[
    { "show":true,
      "icon":"static/images/arch.svg",
      "fallback":"\u2699\uFE0F",
      "url":"services#architecture",
      "label":"" }
  ]
}
```

`label` filled by `deliverables_de.json` overlay.

Services

Purpose: Detailed service cards with descriptions and footer integration.

- Configurable column count (default 3)
- Per-card: anchor, icon, url, `in_footer`
- **`in_footer:true`** → card auto-appears in Footer
- External link → new tab; internal → SPA route
- Dedicated per-component inotify for icon files

```
{ "title_align": "center",
  "columns": 4,
  "card_min_width": 200, "card_max_width": 300,

  "services":[
    { "show":true, "anchor":"architecture",
      "icon":"static/images/arch.svg",
      "url":"/portfolio#abc",
      "in_footer":true,
      "title":"", "description":"" }
  ]
}
```

✔ **Shared dependency:** `shared/ServicesData` is used by both *Services* and *Footer* — single source of truth, no duplication across components.

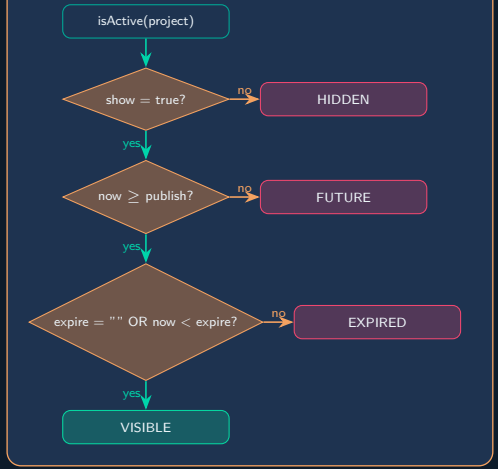
Features

- Configurable N-column grid of project cards
- **Tag-based filtering** — client-side, instant response
- **Time-gated visibility:** publish + expire with full timezone offset support
- Deep-link via anchor field (`#project-name`)
- **Locale overlay matched by `name` key** — not index, so array reordering never causes mismatch
- Live-reload: `portfolio.json` change → instant page update

portfolio.json

```
{ "columns": 3, "card_width": "flexiable",
  "card_min_width": 240, "card_max_width": 340,
  "filter_position": "top", "filter_mode": "nested",
  "show_tags": true, "tags_clickable": true,
  "show_link_icon": false, "explore_label": "",
  "link_icon": "static/images/link.svg",
  "projects":[
  { "name":      "tls-comm-stack", "show": true,
    "publish":   "20260101 09:00 +0100",
    "expire":    "20261231 23:59 +0100",
    "url":       "/blog/tls", "anchor": "tls-comm-stack",
    "tags":      ["C++17","TLS","Automotive"],
    "title":     "", "description":""
  }
]}
```

Time-Gate Logic — isActive()



Architecture

- Posts stored as **Markdown files** with YAML front-matter
- Directory convention: `content/blog/YYYY-MM-DD-slug/{lang}/post.md`
- **Language fallback chain:** requested locale → en → first found
- Two views: **list** (excerpt + meta) and **detail**
- excerpt from front-matter or auto-extracted (first 160 chars of body)
- Shared **TagRegistry** with Portfolio and Article

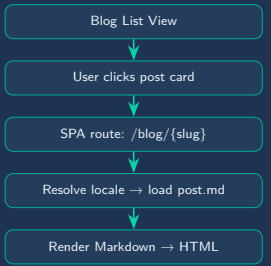
Directory Structure

```
content/blog/
 2024-02-20-cmake-modern/
   en/post.md
   de/post.md
   tr/post.md
 2024-04-02-tls-embedded/
   en/post.md    <- EN only; DE/TR fall back to EN
```

post.md front-matter

```
1  ---
2  title:   Modern CMake Patterns
3  date:    2024-02-20
4  author:  Fehmi Demiralp
5  excerpt: Short summary for list view
6  tags: FFS, Web Server, Internet
7  draft:   false
8  publish: 2024-03-01
9  expire:
10 ---
11
12 ## Introduction
13 Full Markdown content follows here...
```

List to Detail Navigation Flow



Security-First Design

Anti-spam by architecture: email must be verified via **One-Time Password (OTP)** before any message is forwarded.

- Name + Email + Message → **[Send Code]**
- OTP delivered to user's email via SMTP
- User enters OTP → **[Submit]**
- Message forwarded to site-owner's email
- Submissions **persisted** to `messages/` directory
- OTP timeout configurable (default 180 seconds)
- No third-party form service — **self-hosted**, GDPR-clean

app.json — SMTP configuration

```
{ "spam_threshold":    3,
  "min_fill_seconds":  8,
  "smtp": {
    "host":                     "smtp.example.com",
    "port":                     587, "tls": "starttls",
    "username":                 "no-reply@example.com",
    "password":                 "your-password",
    "from_name":                "Your Name",
    "from_address":             "no-reply@example.com",
    "verify_timeout_seconds":   180
  }}
```

OTP Verification Flow

1. Fill name, email, message
2. Click: Send Verification Code — server side
3. OTP arrives in user's inbox
4. Enter OTP code in form
5. Click: Submit — validated
6. Message delivered to owner's email — SMTP
7. Record saved to messages/ dir — persistent

✔ `contact.json` is NOT locale-overlaid — it holds site-owner business data (name, address, phone) which is locale-independent.

About

Unique design: No JSON data file — **100% driven by i18n XML bundles**

- Biography paragraph (`about.bio`)
- **Core Skills** — 6 large cards:
  C++, Embedded, State Machines, TDD, Linux, AUTOSAR
- **Additional Skills** — smaller cards:
  Crypto/TLS, CMake/CI, Qt/QML, Protobuf, SQL, STM32, Real-Time, Bash, OOP, Git/Agile
- Placeholder expansion: `@key@` → value
- **Static page** — no live-reload needed
- All translations in `strings-about{,_de,_tr}.xml`

Testimonials

Purpose: Build trust with prospective clients via rated client testimonials.

- JSON array of testimonial objects
- Per-testimonial: name, role, company, rating (1–5 stars), text
- Locale overlay: **only** `text` is translated; name/company/role are locale-independent
- Star rendering via CSS character entities
- **No live-reload** (editorial / curated content)

```
[{ "name":    "Dr. Marcus Weber",
   "role":    "CTO",
   "company": "AeroDyn Systems GmbH",
   "rating":  5,
   "text":    "English text here."
}]
```

▲ About is the only component that does *not* use JsonDataObject — intentionally lightweight. Content is editorial, not data-driven.

✔ **deepMerge by name-key:** `testimonials_de.json` only needs `name` + `text`. No risk of index mismatch if testimonials are reordered.

core/src/sdk/ — Classes

Class	Purpose
JsonDataObject	Base: load, save, watch, notify
SiteInfo	site.json reader + overlay
Person	contact.json person data
TimeInfo	publish/expire parser + eval
LinkResolver	Internal vs external URL logic
Replacer	@placeholder@ expansion
ReplacerGroup	Chained replacer pipeline
JsonReplacer	JSON-keyed replacement
CaseString	Case-insensitive comparisons
Argv0	Binary path resolution

JsonDataObject subclass pattern

```
class MyData : public JsonDataObject {
public:
  MyData() : JsonDataObject("config/my.json") {}
  ~MyData() override { stopWatch(); } // MUST be first line
protected:
  void fromJson(nlohmann::json const& j) override;
  nlohmann::json toJson() const override;
};
```

shared/ — Cross-Component SDK

Used by 2+ but not all components:

- shared/tags/ → **TagRegistry** used by Blog + Portfolio + Article
- shared/services-data/ → **ServicesData** used by Services + Footer

Placement rule:

- Used by all components → core/src/sdk/
- Used by 2–3 components → shared/
- Used by one component only → inside that component's own src/

GlobalFileWatcher — Scalable Reload

Before: N users → N×4 inotify threads.
After: N users → **1 thread** for entire process.

Build	LIVE_NOTIFY	Behaviour
testmode	defined	Real inotify thread
Release	not defined	All stubs (zero threads)

Part III

Deployment & Value

Container Toolchain ▪ Competitive Positioning ▪ Roadmap

Theme Directory Structure

```
core/static/themes/
  default/css/style.css    <- shared base
  ocean/css/style.css      <- blue palette
  forest/css/style.css     <- green palette
  mistery/css/style.css     <- dark/pink palette

components/services/static/themes/
  default/services.css
  ocean/services.css
  forest/services.css
  mistery/services.css
```

Theme is selected in `site.json`:

```
{ "theme": "ocean" }
```

Layout Control

screen_width	Effect
"restricted"	Max-width container, centred
"full"	Edge-to-edge layout

✔ Adding a new theme = new CSS directory. Zero C++ changes. Zero recompile.

CSS Load Cascade

1. Core `style.css` (shared base) — all themes
2. Component `<n>.css` per page — scoped CSS
3. Project-level overrides — site-specific

Final rendered styles (highest wins)

sync-project.sh — sync CSS + i18n

```
# Sync from source tree into project directory
bash sync-project.sh projects/sandbox

# Modes:
#  --overwrite-favicons   replace project favicons with FFS defaults
#  --dry-run              preview without changes
#  -q, --quiet            quiet (CI/cron use
#  -s, --short            one summary line per step, no file list
```

Security Layers

TLS/HTTPS	`ffscm cert generate` issues project-specific certificates. OpenSSL integration via Wt.
OTP Contact Form	Email verification before any message is forwarded. No bot submissions possible.
XSS Prevention	`HtmlUtils::makePlain` applied to all user-visible strings. No raw HTML injection paths.
Atomic Writes	`.tmp` → `rename()` for all JSON saves. No partial-write corruption.
LIVE_NOTIFY Guard	inotify threads compile out in Release builds. Zero attack surface from file watchers.
Minimal Deps	Only 3 runtime dependencies. No npm supply-chain exposure.

Crash — Resilience Testing (TESTMODE)

13 crash scenarios for container and supervisor validation:

#	Type	Scenario
s1	sig	Null ptr deref → SIGSEGV
s2	sig	Stack buffer overflow
s3	ub	Use-after-free
s4	abort	Pure virtual call
s5	sig	Stack overflow (infinite recursion)
s6	sig	Division by zero
s7	abort	Unhandled exception
s8	abort	std::abort()
s9	exit	std::exit(1)
s10	sig	SIGILL illegal instruction
s11	abort	Double free
s12	abort	std::terminate()
s13	abort	throw from noexcept

✔ `"show":false` in `site.json` keeps Crash invisible to end users but accessible via URL for testers. `PRODUCTION` compile flag removes it entirely.

ffscm Command Reference

Command	Purpose
ffscm rebuild development	Build dev container image
ffscm start development	Start dev container
ffscm attach development	Shell into container
ffscm rebuild production	Build production image
ffscm check production	Full health check
ffscm cert generate	Generate TLS certificates
ffscm logs backup	rsync logs to SSH host

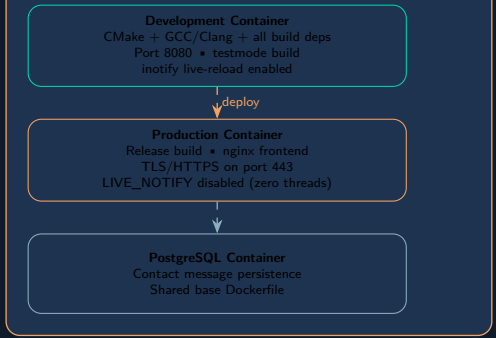
Quick-start workflow

```
# First-time TLS setup
export CERT_PASS='passphrase'
ffscm cert generate sandbox

# Development cycle
ffscm rebuild development
ffscm start   development
ffscm attach  development
# Inside container:
bash rebuild-cmake.sh testmode

# Production deploy
ffscm rebuild production --project serbest --start
```

Two-Container Strategy



Transfer Tools

sync-project.sh	CSS + i18n → project dir
ffs-pack.sh	Pack project for transfer
ffs-untar.sh	Unpack + meld diff review

Build System & Delta Packaging | CMake Modular Build + AI-Assisted Patch System

CMake Build Targets

Target	Type
ffs	Executable (entry point)
libffs-sdk.so	SHARED library
libffs-system.a	STATIC library
app.ffs	SHARED
router.ffs	SHARED
navbar.ffs	SHARED
footer.ffs	SHARED
*.ffs	SHARED (one per component)
crash.ffs	SHARED (TESTMODE only)

Build commands

```
# Full clean rebuild
bash rebuild-cmake.sh testmode

# Single component (fast rebuild)
cmake --build build/testmode \
  --target services

# Run server
bash run-webserver.sh testmode  # port 8080
```

Delta Patch Package Format

Structured .tgz transfers — Claude AI → developer:

```
ffs-{\ffsversion}-navbar-cursor.tgz
  preprocess/          <- git mv/rm before copy
    01-rename.sh
  ffs/                 <- changed files only
    core/src/...
    components/.../
  postprocess/         <- runs after copy
    git-commit-{\ffsversion}.sh
```

Naming convention: v<topic>.<iter>.<fix>-<comment>

- ➜ topic = active roadmap topic number
- ➜ iter = patch count within topic (starts 0)
- ➜ comment = mandatory content description

✔ **Full traceability:** every change is in a named patch. git-commit-v*.sh is manually reviewed before execution. Preprocess handles file renames safely via git.

i18n System — Two Complementary Layers

1. **Wt i18n XML bundles** (`strings-<comp>*.xml`) — UI labels, buttons, form placeholders, navigation display names
2. **Locale-overlay JSON** — content: titles, descriptions, blog text, testimonials

Both layers use the **same resolution priority chain**: URL prefix → cookie → Accept-Language → site.json default

URL Language Routing

URL	Language served
/	default_language
/de/services	German
/tr/about	Turkish
/en/blog	English

site.json — language configuration

```
1  {
2    "default_language": "de",
3    "languages": ["de", "tr", "en"]
4  }
```

Menu Label Localisation

```
1  // site_de.json
2  { "pages": [
3    {"name":"sliding",  "display":"Startseite"},
4    {"name":"services", "display":"Leistungen"},
5    {"name":"blog",     "display":"Blog"},
6    {"name":"contact",  "display":"Kontakt"}
7  ]}
```

Locale-file page order takes precedence over base. Base pages not in locale file are appended last. Matching is by `name` key — **no index shifting**.

✔ Adding **French:** add `strings-*_fr.xml` + `*_fr.json` files + append `"fr"` to the `languages[]` array. **Zero C++ changes. Zero recompile.**

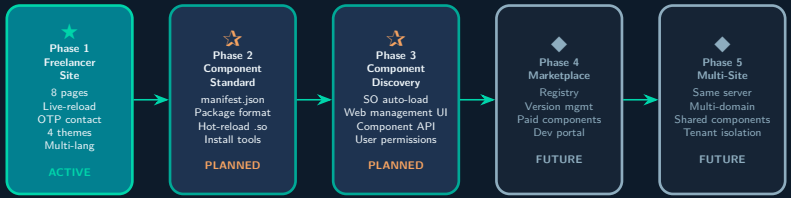
Feature	FFS	WordPress	Next.js	Hugo	Wagtail	Wix
Language	C++23	PHP	JS/TS	Go	Python	SaaS
Component model	.so live	plugin	module	partial	app	widget
Guaranteed Server-side Render	✔	✘	✘	✔	✘	✘
Live content reload	✔	✘	✔	✘	✘	✔
OTP contact form	✔	plugin	custom	✘	plugin	plugin
Publish scheduling	✔	✔	custom	✘	✔	✔
Multi-language	✔	plugin	i18n lib	✔	✔	✔
Self-hostable	✔	✔	✔	✔	✔	✘
Supply-chain risk	3 deps	500+	500+	~50	200+	N/A
RAM idle	~8 MB	30 MB	60 MB	5 MB	50 MB	N/A
Build artifact	binary	runtime	bundle	static	runtime	cloud

✔ **FFS wins on:** Performance, security surface, dependency hygiene, C++ control

▲ **Trade-offs:** Smaller ecosystem, C++ expertise required, no GUI admin panel (yet)

Ideal for: C++ shops, high-security environments, embedded/edge deployment

5-Phase Roadmap | From Freelancer Platform to Component Marketplace



✔ **Phase 1 is production-ready:** All 8 page components operational, container toolchain complete, multi-language live, HTTPS/TLS, OTP contact, 4 themes.

Phase 2 key deliverable: `manifest.json` standard + hot-swap of `.so` files without server restart — the *true* zero-downtime plugin architecture.

Business Value

- ✔ **Content updates without developer:** JSON + Markdown editable by any team member. No CMS admin-panel attack surface.
- ✔ **Scheduled content:** publish/expire per slide or portfolio project. No midnight deployments.
- ✔ **Multi-market from one deployment:** EN/DE/TR with per-market URLs (`/de/`, `/tr/`).
- ✔ **GDPR-safe lead capture:** OTP-verified contact form. No third-party form services.
- ✔ **No vendor lock-in:** self-hosted C++ binary. Full infrastructure ownership.

CTO / Engineering Manager Value

- ✔ **Component isolation:** add or remove pages without touching the core. Zero regression risk.
- ✔ **Deterministic builds:** binary artifact, identical behaviour dev to prod. No dependency drift.
- ✔ **Auditable dependencies:** 3 libraries with known licenses (GPL/Commercial, BSL-1.0, MIT).
- ✔ **Performance:** ~8 MB RAM idle, native code, scales on commodity hardware.
- ✔ **Future-proof architecture:** clear five-phase roadmap on solid C++23 foundations.

✔ **Bottom line:** FFS is not just a website builder — it is a **production-grade, maintainable platform** built on C++23 best practices, with a clear component contract and a five-phase growth roadmap. Every new component added builds on the same solid foundation.

Known Constraints & Mitigations | Transparent Risk Assessment

★ Current Constraints

✘ No GUI admin panel	Content editing requires direct JSON/Markdown file access (SSH, git). No browser editor.
✘ C++ expertise required	Component development needs C++23 knowledge. Not a no-code platform.
✘ No hot-swap in production	Adding a component requires a server restart. (Phase 2 target: fix this.)
✘ Small ecosystem	No community marketplace yet. Components are bespoke. (Phase 4 target.)
✘ No WYSIWYG	Blog and portfolio updates are file-based.

★ Mitigations & Roadmap Answers

✔ Git-based workflow	PR + review for content changes. Better audit trail than any CMS.
✔ JSON is readable	Non-engineers can safely edit `services.json` with basic training.
✔ Phase 2 plans hot-swap	Manifest-based component discovery and `.so` hot-reload targeted for Q3 2026.
✔ Phase 4 adds marketplace	Registry and versioned component distribution planned for 2027.
✔ Container restart is fast	Docker/Podman restart under 5 seconds — viable until Phase 2 arrives.

Questions & Next Steps | Open Discussion

Common Decision-Maker Questions

- **How long to add a new component?** Days for a full page: C++ source, JSON config, CSS (4 themes), i18n XML, tests.
- **Can non-C++ devs maintain content?** Yes. JSON + Markdown via git. No C++ required for content work.
- **What is the upgrade path?** Delta TGZ patches, reviewed via meld + git. Fully incremental and auditable.
- **GDPR compliance for contact form?** OTP-verified, self-hosted, no third-party data transfer.
- **Licensing?** Wt: GPL or Commercial. Boost: BSL-1.0. nlohmann/json: MIT.

Immediate Next Steps

1. Explore live demo: `http://87.123.247.235:9980/`
2. Identify which components match your site requirements
3. Choose a project name → initialise `projects/<n>/config/`
4. Run `ffscm rebuild development` to validate build environment
5. Define Phase 2 requirements for hot-swap component loading

✔ Full documentation available in source tree:
`README.md` • `CLAUDE.md` • `CLAUDE-components.md`
`DEPENDENCIES.md` • `FFS-PACKAGING.md`

Thank You

Foundational Feature Server

v0.63 ▪ Core Architecture ▪ Components ▪ C++23 ▪ Wt 4.12

Live Demo https://fedem.eu

Docs: https://fedem.eu/ffsdocs

Fehmi Demiralp
demiralp@fedem.eu