ARCBOUND

INTERNAL PLANNING BRIEF

WORKFLOW · BUILD SPEC · SOP / DRAFT V0.3

ArcOS → Web

Turning a finished discovery into a real website. From brand DNA, sitemap, and copy to a production WordPress build that *loads in a quarter second* and scores a hundred.

DOCUMENT
ArcOS Web Generation

OWNER
Eli Wright

BUILDING WITH
Anton Langer · Raphael (Baseplate)

SOURCE
Web sync · SOP · gen spec

STATUS
Draft — for review

DATE
June 08, 2026



01 WHY THIS EXISTS

The objective

ArcOS already runs discovery, brand DNA, the visual kit, and the content engine. The missing piece is the last mile: turning that output into a site.

The goal of this workflow is simple. Once a client's visual identity is locked and the sitemap and copy are organized, we push a button and the system generates the website in modular, performance-ready pieces that Anton and Irina drop straight into our theme. The system gets it to *eighty or ninety percent*. Then we add the human taste that keeps it from looking vibe-coded, and that's the part clients are actually paying for.

NORTH STAR

A site that feels custom, loads in a quarter second, and the client can take anywhere.

Custom-feeling, not templated. Fast enough to win on performance alone. Portable enough that there's no lock-in to sell against.

02 THE CONSTRAINT THAT SHAPES EVERYTHING

WordPress, on purpose

Tim's call, and it's the right one: the build stays platform-agnostic as long as we feed it the right inputs. WordPress is home base. Lovable, Vercel, Figma, and Claude become design tools that produce HTML we pull down. They are never the platform the client gets stuck inside.

A Compartmentalization

It's one of the only content management systems that lets us strip the framework to nothing and load a component's code **only when that component is used**. That is the single thing that gets a site from eighty to a hundred. No other builder allows it cleanly.

B Portability

The client owns the site and can move it anywhere with no re-coding and no penalty. That is a promise Lovable can't make, and it's the heart of the WordPress push right now: *democracy of choice* for the client.

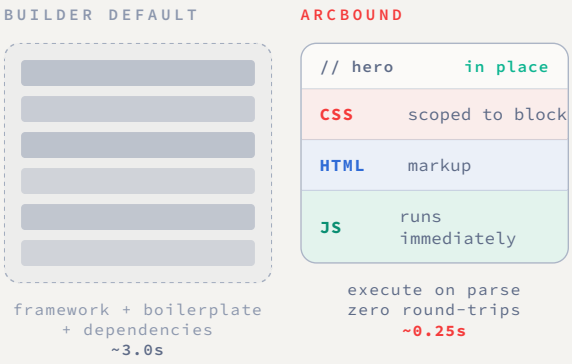
03 THE GOLD STANDARD — THIS IS THE MOAT

The performance principle

Every builder, Lovable included, builds off its own code base. That code base always ships boilerplate into the page. You can get close, fifty to eighty, the way we did testing Arcbound. Getting from *eighty to a hundred* is the next echelon, and that's where we live.

Here's how. Strip the entire framework and load nothing by default. Then each section ships as one self-contained block: its CSS sits right above the HTML, the JS right below it. When the browser parses that block it executes it in place. No round-trip to the server for a dependency. No extraneous code to wade through. Roughly three seconds of load time collapses to a quarter second.

Speed is the whole point. Faster means lower bounce, longer sessions, and it's how you get found now: SEO, AEO, AI overviews, and agents all reward it. Fast reads as trust and authority.



3.0s → 0.25s
LOAD TIME PER PAGE

80 → 100
PAGESPEED CEILING

0
EXTRANEOUS DEPENDENCIES

This is the moat. Not just that we're fast, but that every client lands at a performance score they can't reach on their own, and we stay on the cutting edge of it for them.

03.1 THE THEME ANTON SHARED ALREADY PROVES IT

The theme is the spec

We don't have to theorize about the format. The theme files lay out exactly what the generator has to hit. Everything ArcOS outputs has to drop into this structure cleanly, so the build stays as fast as the principle promises. The pages that follow break the system down end to end: the component library, the anatomy of a file, the tokens, the runtime contracts, and the bar a section has to clear before it ships.

- **Code lives with the component.** Each section ships its CSS inline, directly above its markup and scoped to the section class; any JavaScript rides a guarded `wp_footer` function so it loads only where the section is used. Nothing global leaks.
  `home-banner.php, home-about.php, internal-banner.php`
- **Sections are wired by shortcode.** A page only renders the sections it actually uses, so unused code never loads.
  `shortcodes.php — home_banner, home_services, home_about, internal_banner …`
- **The fat is stripped.** Emoji scripts removed, block-library CSS dequeued, srcset off, native lazy-load off. `functions.php`
- **Loading is hand-built and lean.** A custom IntersectionObserver swaps `data-url → src` about 150px before viewport, so nothing loads early. `theme-changes.php`
- **Reveal animations are already there.** `.fade-from-left / -right / -bottom` with an observer adding `.active`. The generator should output these hooks, not reinvent them. `theme-styles.php`

WHAT A GENERATED SECTION MUST LOOK LIKE

One block, in order, every time:

1. scoped `<style>` directly above
2. the section HTML
3. its `<script>` directly below

Mobile breakpoints and collapse behavior baked in from the theme's boilerplate, not improvised.

BRAND PARAMETERS TO FEED IN

- color system
- type scale
- spacing / ma
- reveal hooks
- SEO rules
- micro-animation

03.2 WHAT THE GENERATOR TARGETS

The component library

The theme ships nineteen sections, each already wired by shortcode in `shortcodes.php`. ArcOS doesn't invent components, it fills these. Two families: the homepage set and the internal-page set. Banners use the inline style pattern; everything else uses the guarded footer pattern.

HOMEPAGE 13

`home_banner`	Hero, first view	INLINE
`home_about`	Who we are	FOOTER
`home_services`	What we offer	FOOTER
`home_results`	Outcomes in numbers	FOOTER
`home_portfolio`	Selected work	FOOTER
`home_testimonials`	Social proof	FOOTER
`home_team`	The people	FOOTER
`home_awards`	Recognition	FOOTER
`home_mission`	Purpose, values	FOOTER
`home_news`	Latest posts	FOOTER
`home_social`	Social feed	FOOTER
`home_cta`	Conversion band	FOOTER
`home_contact`	Get in touch	FOOTER

INTERNAL PAGES 6

`internal_banner`	Page hero + H1	INLINE
`internal_content`	Body, rich text	FOOTER
`internal_fullwidth`	Edge-to-edge block	FOOTER
`internal_services`	Service detail	FOOTER
`internal_cta`	Conversion band	FOOTER
`internal_contact`	Contact block	FOOTER

THE SET IS THE CONTRACT

Every shortcode resolves to a file in `template-parts/sections/`. Generation maps a client's content to these sections, and only these. New section types are a theme decision, not something the generator improvises.

03.3 PART II · HOW EVERY FILE IS BUILT

Anatomy & the 1:1:1 rule

The slug is the single source of truth. One slug drives the file name, the root CSS class, and the shortcode. Every section then follows the same top-to-bottom order, so the generator can produce it the same way every time.

ELEMENT	PATTERN	EXAMPLE
File name	{slug}.php	home-banner.php
Root CSS class	.{slug}	.home-banner
Shortcode	{slug}, hyphen → underscore	home_banner

Home sections live in `template-parts/sections/home/`, internal in `.../internal/`.

```
// home-banner.php                                                    canonical order

<?php $atts = shortcode_atts(array(           // 1  attributes, always first
    'title' => '',
), $atts); ?>

<style type="text/css">                       // 2  CSS, scoped to .home-banner
  .home-banner h1{ font-size:var(--xl); color:var(--color-1); }
  @media(max-width:1000px){ /* tablet */ }  @media(max-width:750px){ /* mobile */ }
</style>

<section class="home-banner section-padding">          // 3  HTML
  <div class="content-width">
    <h1 class="home-banner__title fade-from-bottom"><?= $atts['title']; ?></h1>
  </div>
</section>

<?php if(!function_exists('apply_home_banner_styles')){   // 4  JS via wp_footer
    function apply_home_banner_styles(){ ?>
        <script>window.addEventListener('load',function(){ /* scoped JS, omit if none */ });</script>
    <?php } add_action('wp_footer','apply_home_banner_styles'); } ?>
```

```
// shortcodes.php                                                     registration

function home_banner($atts, $content=null){
    ob_start();
    include(locate_template('.../home/home-banner.php'));
    return ob_get_clean();
}
add_shortcode('home_banner','home_banner');
```

1. **Attributes first** — always a `shortcode_atts()` block, even if empty.
2. **CSS inline, scoped** — every selector under `.{slug}`, tokens not hardcoded, 1000 / 750 breakpoints.
3. **HTML** — `.section-padding` on the section, `.content-width` inside, BEM subclasses.
4. **JS last, in wp_footer** — guarded, `window load`, omitted entirely if the section has none.

03.4 WHAT THE GENERATOR CONSUMES

Tokens & primitives

The system reads from `:root` and the layout primitives, and never invents new ones. The theme ships placeholder tokens from the 2022 base. ArcOS overwrites them with the client's brand before generating.

COLOR — placeholder, per-client



--color-1
#7d50df



--color-2
#7d50df



--color-3
#ffcb05

Overwritten with the client palette at generation time.

TYPE SCALE

--xl	55px · h1
--lg	40px · h2
--md	30px · h3
--sm	20px · h4
--default	16px · body

Steps down at 1000px and 750px.

FONTS — placeholder, per-client

--heading-font	Playfair Display
--default-font	Montserrat
--accent-font	Montserrat

Swapped to the brand stack at generation time.

LAYOUT PRIMITIVES

.content-width	1230px, centered
.section-padding	min(150px, max(5%,50px))
.button	token-driven, hover --color-1
.absolute-cover	full-bleed media

THE RULE If a token exists, use it. No hardcoded hex or px where `:root` has a value. Headings already map to the scale, so a generated `h1` inherits `--xl` and steps down on its own. The generator sizes by token, not by guess.

03.5 TWO CONVENTIONS THE OUTPUT MUST HONOR

Runtime contracts

Two behaviors are already wired into the theme through observers in `theme-changes.php`. Generated markup hooks into them. It never reimplements them.

Lazy loading

Media doesn't load until it's near the viewport. `srcset`, native lazy-load, and the decoding attribute are all stripped in `functions.php`.

markup

```
<img data-url="/img/x.jpg"
     width="800" height="600" alt="…">

<span class="absolute-cover"
   data-bg-img="/img/bg.jpg"></span>
```

- Images and iframes use **data-url**; backgrounds use **data-bg-img**.
- The observer swaps the real source about **150px before** the viewport, then stops watching.
- Always set width and height so nothing shifts as it loads.

Reveal animations

Elements animate in on scroll through three hooks. An observer adds `.active` at 15% visibility, once.

markup

```
<div class="fade-from-bottom">
  …
</div>
```

- Three directions: **.fade-from-left**, **-right**, **-bottom**. Pick one per layout.
- Disabled under **750px**; mobile renders elements visible.
- Don't hand-roll scroll JS for simple reveals.

Both observers run on load. The generator's only job is to emit the right attributes and classes, and the theme handles the timing, the thresholds, and the cleanup.

03.6 WHAT GOOD OUTPUT LOOKS LIKE

From shell to section

The same file, before and after. The shell is what Anton sent. The generated version is a themed `home_banner` the system should produce. Everything highlighted is a theme hook, not invented code.

// home-banner.php — shell

```
<style>

</style>

<section class="home-banner section-padding">
  <div class="content-width">

  </div>
</section>
```

// home-banner.php — generated

```
<style>
  .home-banner{ position:relative; color:#fff; }
  .home-banner h1{ font-size:var(--xl); margin:0 0 .4em; }
  .home-banner .lede{ font-size:var(--sm); max-width:34ch; }
</style>

<section class="home-banner section-padding">
  <span class="absolute-cover" data-bg-img="/img/hero.jpg"></span>
  <div class="content-width">
    <h1 class="home-banner__title fade-from-bottom">Your headline</h1>
    <p class="lede fade-from-bottom">One-line subhead.</p>
    <a class="button" href="/contact">Get started</a>
  </div>
</section>
```

NOTICE

- var(--xl) · sized by token
- .content-width kept
- data-bg-img · lazy
- .absolute-cover
- .fade-from-bottom · reveal
- .button · themed CTA
- CSS scoped under .home-banner
- no JS → footer fn omitted

03.7 PART II · THE GATE BEFORE A SECTION SHIPS

Generation checklist

The canonical checklist from the generation spec. Every section, human or AI, is verified against all of it before Anton and Irina place the file.

- Slug matches the **file name, root class, and shortcode** (the 1:1:1 rule).
- Shortcode is registered in **shortcodes.php**, not inside the template file.
- File order is **attributes → style → HTML → JS**, and nothing else.
- Every CSS selector is scoped under **.{slug}**; zero global selectors.
- No hardcoded colors; all use **var(--color-*)** or explicit neutrals.
- No hardcoded font sizes; all use **var(--xl)** through **var(--default)**.
- Fonts via **var(--heading-font)**, **--default-font**, **--accent-font**.
- Responsive overrides included at **1000px and 750px**.
- Wrappers used: **.section-padding** on the section, **.content-width** inside, **.absolute-cover** for full-bleed media.
- Lazy media uses **data-url**, backgrounds use **data-bg-img**; no src, srcset, or inline lazy/decoding attrs.
- Reveals use **.fade-from-***; no custom animation logic in the section.
- JS wrapped in **apply_{slug}_styles()** with a function_exists guard; **window load**, not document.ready.
- **No JS** block at all if the section isn't interactive, and nothing enqueued in the template.

PER-SECTION FLOW

01	→	02	→	03	→	04	→	05
Inputs		**Generate**		**Format**		**Checklist**		**Integrate**
tokens + content + contract		CSS · HTML · JS		1:1:1, file order		verify every item		place file · shortcode wired

04 END TO END

The workflow

Six stages, left to right through the system. Nothing downstream starts until the inputs above it are locked. This loop is what feeds Phase 1 and Phase 2 of the build SOP in Part III; Phases 3 to 5 carry it to launch.

01

Inputs assembled IN ARCOS

Discovery done. Visual identity locked (brand DNA + kit), sitemap approved, content organized and written, whether by AI with human input or by hand. The button does nothing until these are set.

02

Generate, section by section KEY DECISION

Don't generate the whole page at once. Build it one section at a time so the model spends full capacity on each piece instead of spreading thin across a homepage. Three hero options, then the body, then about, then services. Feed it the theme plus the brand parameters plus the SEO rules.

03

Theme-match the output

Output lands in the compartmentalized format: scoped CSS above, HTML, JS below, with the theme's breakpoints and mobile collapse built in. First-pass SEO to roughly fifty percent on structure and headings. Meta titles and descriptions queued for later.

04

Human taste THE PART THAT MATTERS

Apply the custom touch points, fix the granular thematic bits, and kill anything that reads vibe-coded. This is where Eli, Anton, and Tim turn a good draft into something that looks deliberately designed for the person.

05

Dev integration ANTON + IRINA

Extract the modules and plug them into the theme. On a new site it's close to plug-and-go. The modular pieces double as a feedback loop: drop a section on staging, show it, and feed the live thing back into the system for the interior pages.

06

SEO + QA pass

AI runs the QA checklist and finishes the SEO lift per page. Meta titles and descriptions push per-site through the WordPress API with that site's credentials. Anton's SEO and AEO diligence layers on top.

05 WE BUILD BOTH, THEN PICK

Two tracks to test

This is uncharted territory, so we run an A/B and let the work decide. Whichever produces clean, theme-ready modules with the least cleanup and the fastest path to staging wins.

A Native HTML

HOW

ArcOS generates straight HTML, matched to the theme as closely as the parameters allow, then refined. Claude is strongest here.

WHY IT MIGHT WIN

Closest to final output, fewest translation layers. Daphne's site pages were done this way and they worked.

OPEN QUESTION

How tightly we can hold it to the compartmentalized chunk format instead of it drifting on the thematic details.

B Figma bridge

HOW

Generate the visuals into Figma as modular pieces, then pull them down to HTML through the Figma-to-HTML plugin.

WHY IT MIGHT WIN

A stronger, more art-directed visual layout from a real design surface before anything touches code.

OPEN QUESTION — VALIDATE FIRST

Can Figma export as compartmentalized chunks (header / about / services as separate module files) rather than one full-page HTML dump? Plugin quality is unproven. This is the blocker.

DECISION RULE

Likely a hybrid: Figma for visual exploration, native HTML for the build. We commit to whichever gives us theme-ready modules with the least manual cleanup and the fastest time-to-staging. Anton has no strong lean, so the test settles it.

06 WHAT'S STILL MANUAL OR UNSOLVED

Known gaps

- **There is no theme API.** WordPress's API only touches posts, blogs, and meta titles and descriptions, not the theme, and the theme is where all our section code lives. Integration stays semi-manual: someone pushes the buttons. Page creation and meta can be API-driven per site; theme code is hand-placed (fast, but by hand).
- **Established sites are harder.** A new build is "go create this." A live site like Dr. Angel, with existing pages and posts, means managing URLs and duplicates so nothing breaks. Same modular pull-down, but migration and page creation stay manual for now.
- **Hosting automation is low value right now.** Standing up staging on Hostinger takes about five minutes and pennies, so an API for it isn't worth building yet. Revisit only if it removes real friction.
- **A future theme-level API is possible** and could let us manipulate the theme directly. If we build it, it has to be locked down. Security first, per Anton.

07 WHY A CLIENT PAYS FOR THIS

The pitch

This is what the workflow unlocks commercially, and it feeds Arcbase plus recurring site maintenance. The line is: you can DIY a site with Lovable, but here's what you get on top of AI.

01 · TASTE

AI-assisted, human-finished. The custom touch a builder can't give you.

02 · PERFORMANCE

Scores you won't hit on your own, and we hold you there over time.

03 · PORTABILITY

You own the site and can take it anywhere. No re-coding, no lock-in.

And it's recurring. Site maintenance becomes a subscription rather than a one-off, because we keep the site on the cutting edge of SEO and AEO for them. That's the human layer on top of every AI-assisted step.

III PART III · HOW A SITE ACTUALLY SHIPS

The build & launch SOP

Generation produces the sections. The SOP is the end-to-end process those sections drop into, from staging to post-launch. Five phases, the same on every project, so quality and handoff stay consistent.

PHASE 1

Setup & homepage

Staging, install, migrate, build header / home / footer.

PHASE 2

Internal pages

Remaining templates, then a second migration.

PHASE 3

Pre-launch QA

Updates, forms, SEO audit, performance, cleanup.

PHASE 4

Launch

Environment-dependent go-live.

PHASE 5

Post-launch

SMTP, SSL, monitoring, client handoff.

AI PLUGS IN

Generation feeds **Phase 1 (homepage sections)** and **Phase 2 (internal sections)** — the design-to-code step. Migration, QA, launch, and handoff stay human-run, with AI assisting on the SEO and QA passes. The theme has no API yet, so placing code is still hands-on.

TOOLING STACK

HostGator · arcbounddev.com/{client} · Migration Guru · Infinite WP · Yoast SEO · Screaming Frog · Redirection · Post SMTP · PageSpeed Insights

P1 PHASE 1 · ENVIRONMENT + FIRST BUILD

Setup & homepage

Stand up the staging environment, get content in, and build the header, homepage, and footer against the theme.

1.1 HOSTING & DATABASE

- Create a new **database** in HostGator.
- Create a **DB user** and assign it to that database.
- Create the staging directory `arcbounddev.com/{client-name}` — lowercase, hyphens.

1.2 WORDPRESS INSTALL

- Download the latest WordPress from wordpress.org.
- Upload all files to the staging directory via FTP.
- Run the install wizard and connect the new database.

1.3 CONTENT MIGRATION

- **Existing WP:** Migration Guru full copy to staging; verify content, media, settings.
- **Non-WP:** fresh install, recreate pages, copy all titles + meta descriptions.

1.4 CONFIGURATION

- Add staging logins to **Infinite WP**.
- Yoast: images **forward to themselves**.
- Yoast: **noindex** archive pages (authors, categories, tags).
- Run the full site startup procedure.

1.5 THEME: HEADER, HOME, FOOTER

- Global CSS in **theme-styles.php** (via wp_head); edit **:root** vars per client, never hardcode.
- Header in **header.php** — CSS-only hamburger, no JS.
- Footer scripts via **wp_footer** through includes/theme-changes.php.
- Home sections are template-parts called as `[home_*]` shortcodes.
- Build header, home sections, and footer to the design.

Lazy load is custom. Image `src` is output as `data-url` and swapped by the observer. Native WP lazy load is intentionally disabled.

P2 PHASE 2 · EVERYTHING ELSE

Internal pages

Build out the remaining page templates and content, then run a second migration to capture anything the client changed on the live site during the build.

2.1 INTERNAL TEMPLATES

- Sections live in `.../internal/`, registered as `[internal_*]`.
- Available: banner, content, fullwidth, cta, contact, services.
- Add a new one by creating the PHP file and registering its shortcode.

2.2 COMPLETE THE BUILD

- Develop all internal pages, sections, and custom functionality.
- Apply scroll reveals with **.fade-from-left / -right / -bottom**.
- Animations auto-disable on mobile (≤750px).

2.3 SECOND MIGRATION

- Re-run Migration Guru (WP→WP) or reconcile content by hand.
- Verify no development work was overwritten.
- Confirm all media is present and paths are correct.

WHERE GENERATED SECTIONS LAND

Phases 1.5 and 2 are the integration point. A generated section comes out as one self-contained module, and the dev step is short:

- Extract the module from the generation output.
- Drop the file into the right `template-parts` folder.
- Confirm the shortcode is registered (already wired in `shortcodes.php`).
- Place the shortcode on the page, then refine the granular thematic bits.

Established sites are the harder case: existing pages and posts mean watching URLs and duplicates so nothing breaks. The modular pull-down still applies; page creation stays manual.

P3 PHASE 3 · QUALITY ASSURANCE BEFORE GO-LIVE

Pre-launch checklist

Complete every item before marking the site ready. Use **Screaming Frog** for the URL and metadata audits.

3.1 UPDATES & PLUGINS

- Themes, plugins, and WP core on the latest version; auto-updates on.
- Required active: Yoast SEO, Forms, Redirection.
- Remove unneeded backend / MU plugins.

3.2 FORMS & FUNCTIONALITY

- Submissions route to the correct recipients.
- Thank-you pages configured and working.
- All buttons link correctly — no placeholders.

3.3 SEO AUDIT · SCREAMING FROG

- Staging URLs match the intended live structure.
- Meta descriptions, titles, and H1s match live ↔ staging.
- Archives set to noindex; image alt + Yoast forwarding correct.

3.4 PERFORMANCE & ACCESSIBILITY

- Site speed acceptable; take a **before** PageSpeed snapshot.
- Full-page screenshot if replacing an existing site.
- Accessibility score green; SEO score green.

3.5 TECHNICAL CLEANUP

- Tracking and analytics added (GA, GTM).
- Test and dev user accounts removed.
- Custom **404.php** in place and styled to match.

Snapshot everything. The pre-launch PageSpeed score and screenshot are the baseline you compare against in Phase 5. Capture them before go-live, not after.

P4-5 PHASE 4 & 5 · GO-LIVE AND AFTER

Launch & post-launch

P4 LAUNCH

Environment-dependent, so it varies by client. Confirm before starting:

- Domain registrar and **DNS access** confirmed.
- Hosting environment decided (HostGator, migrating, client-managed).
- **SSL** plan confirmed (existing, new, auto-provisioned).
- Client-approved **go-live date** and maintenance window set.
- Old-URL **redirects** documented in the Redirection plugin.

Document the specific launch steps per client in the project file — they differ significantly by environment.

P5 POST-LAUNCH

- **Post SMTP** configured and tested; forms send from production.
- HTTPS active on every page; **no mixed content**.
- Site added to **Infinite WP** and to monitoring.
- PageSpeed snapshot taken; compare to the pre-launch baseline.
- New-site screenshot for records.
- Client notified live; confirm **WP admin, hosting, domain** access.
- Provide documentation or training.

Handoff email. Live URL, a summary of what was completed, and next-step recommendations (ongoing maintenance, SEO monitoring).

REF APPENDIX · FILE MAP

Theme structure

Where each responsibility lives. When a client's brand changes, only the `:root` block in theme-styles.php needs editing.

`functions.php`	Theme bootstrap: enqueues styles and fonts, registers menus, disables emojis and block-library CSS, disables WP srcset and lazy load (custom lazy load used instead).
`theme-styles.php`	All global CSS, injected via wp_head. Holds the :root variables for colors and fonts (edit per client) plus layout, menu, button, animation, and responsive styles.
`shortcodes.php`	Registers every section PHP file as a shortcode, for both home (home_*) and internal (internal_*) templates.
`includes/theme-changes.php`	Output via wp_footer. The IntersectionObserver lazy-load script (swaps data-url to src) and the scroll-animation activation script.
`header.php`	Outputs <head>, opens <body>, renders .masthead with logo and the CSS-only mobile nav, opens #main and #main-content.
`footer.php`	Closes #main-content and #main, then triggers wp_footer (which fires every section's JS).
`template-parts/sections/home/`	Home page section partials, included via [home_*] shortcodes.
`template-parts/sections/internal/`	Internal page section partials, included via [internal_*] shortcodes.
`404.php`	Custom 404 error page template.
`style.css`	Theme registration file only — no styles here; use theme-styles.php.

08 OWNERS, TIMELINE, OPEN QUESTIONS

Next steps

Eli CREATIVE · PRODUCT	Build this brief and the workflow map (done). Prompt ArcOS to generate. Stand up both tracks. Pilot on a real client — Jennifer Maanavi is the candidate.
Anton WEB / THEME	Share the SEO research. Run SEO and AEO diligence. Refine thematic granularity on generated output. Scope a secure path for a theme-level API.
Raphael BASEPLATE	Gets the recording and this brief. Builds alongside Eli on the generation pipeline inside ArcOS.
Anton & Irina INTEGRATION	Module extraction and theme integration as each track comes online. They are the fastest path from generated chunk to live page.

TIMELINE Turn around a first generated homepage in a few weeks and test it on a live client.

OPEN QUESTIONS TO SETTLE

01 Can the generator reliably output compartmentalized chunks (CSS above, HTML, JS below) that match the theme?

02 Native HTML or Figma — which wins on quality and time-to-staging, or is it a hybrid?

03 Cleanest path for established-site migration without breaking URLs or creating duplicates?

04 What's the security model for any future theme-level API?

05 How finished should generation be before human refinement — full page, or piece-by-piece with a staging feedback loop?