The Formery — Flexible Page Builder Technical Specification

---

1. Overview

This spec defines the implementation of a flexible, component-based page builder for theformery.com, built on Craft 5's native **nested entries** feature. Content editors will assemble pages by adding, reordering, and configuring a fixed library of components inside a single Matrix field, rather than relying on rigid, single-purpose templates.

1.1 Goals

- Give editors a self-serve way to build new page layouts without developer involvement.
- Keep the component library tight and reusable (7 component types, listed below).
- Reuse shared building blocks (e.g. CTA, image transforms) across components to avoid duplicated logic.
- Integrate the existing Wheelform plugin for the Form component.

1.2 Why nested entries (Craft 5)

Craft 5 replaced the old Matrix "block type" model with **nested entries**: each block type in a Matrix field is a full Entry Type with its own field layout, conditional fields, and (optionally) further nested Matrix fields inside it. This is the recommended approach for page builders in Craft 5 and is what this spec is built around, rather than the legacy Matrix or a third-party builder plugin (e.g. Neo).

---

2. Architecture

2.1 Field structure

Element	Type	Notes
`pageBuilder`	Matrix field (nested entries)	Attached to the field layout of the "Page" entry type (and any other section that needs flexible content)

Element	Type	Notes
Entry Types (one per component)	Nested Entry Type	Each has its own field layout; see Section 4

Editors add one or more entries into `pageBuilder`, choose a component type, fill in its fields, and reorder via drag-and-drop — standard Matrix UX in the Craft control panel.

2.2 Shared/reusable fields

To avoid rebuilding the same fields per component, define these **once** and reuse them across multiple field layouts:

Field handle	Field type	Used in
`ctaLabel`	Plain Text	Header Banner, 1-Col, Banner Text
`ctaLink`	Link field (Craft 5 native `Link` field type — supports URL, entry, email, phone)	Header Banner, 1-Col, Banner Text
`heroImage` / `bannerImage`	Assets (single, restricted to Images volume)	Header Banner, Banner Text

Because Craft field layouts let the same field be added to multiple entry types, `ctaLabel` + `ctaLink` should be built once and marked **not required** everywhere they're used — this gives you the "CTA (Optional)" behaviour for free, and a single place to change CTA logic later (e.g. adding a button-style dropdown).

2.3 Conditional fields

Craft 5's field layout **conditions** (show/hide a field based on another field's value) are used for:

- 1-Column Layout: toggling between Text and Image
- 2-Column Layout: toggling each column's content based on the selected layout variant

This avoids needing separate entry types per variant.

2.4 Template rendering pattern

{# _entry.twig or wherever pageBuilder is output #}

{% for block in entry.pageBuilder.all() %}

```
    {% include "_components/" ~ block.type.handle ~ ".twig" with { block: block } %}
{% endfor %}
```

Each component gets its own partial in `templates/_components/`, named after the entry type handle (e.g. `headerBanner.twig`, `accordion.twig`). This keeps templates isolated, testable, and easy to extend when a new component is added later.

---

3. Component Library

Seven components, as follows.

3.1 Header Banner

Entry type handle: `headerBanner`

Field	Type	Required	Notes
Title	Plain Text	Yes	
Background Image	Assets (single)	Yes	Use a dedicated `heroBanner` image transform (see 5.)
Description	CKEditor (limited config, see 4.1)	No	
CTA Label / CTA Link	Shared CTA fields	No	

3.2 1-Column Layout

Entry type handle: `oneColumn`

Field	Type	Required	Notes
Content Type	Radio Buttons / Dropdown: `Text`, `Image`	Yes	Drives conditional display below
Text	CKEditor (simple config)	Conditional — shown if Content Type = Text	
Image	Assets (single)	Conditional — shown if Content Type = Image	
CTA Label / CTA Link	Shared CTA fields	No	

3.3 2-Column Layout

Entry type handle: `twoColumn`

Field	Type	Required	Notes
Layout Variant	Dropdown: `Text/Text`, `Image/Image`, `Text/Image`	Yes	Drives conditions on both columns
Column 1 Text	CKEditor (limited config, see 4.1)	Conditional — variant is Text/Text or Text/Image	
Column 1 Image	Assets (single)	Conditional — variant is Image/Image	
Column 2 Text	CKEditor (limited config)	Conditional — variant is Text/Text	
Column 2 Image	Assets (single)	Conditional — variant is Image/Image or Text/Image	
CTA Label / CTA Link	Shared CTA fields	No	**Confirmed:** optional CTA button added to this component

Stacks to a single column on mobile (see 4.2 for breakpoint approach).

> Note: this gives editors exactly the three combinations without needing three separate entry types. If more combinations are ever needed (e.g. Image/Text, the mirror of Text/Image), this structure extends cleanly — just add the variant to the dropdown.

3.4 Accordion

Entry type handle: `accordion`

Field	Type	Required	Notes
Accordion Items	Nested Matrix field (nested inside this entry type)	Yes, min 1	Each item = its own mini entry type `accordionItem` with **Heading** (Plain Text) and **Content** (CKEditor)

Front end: render with native `<details>`/`<summary>` where possible for built-in accessibility and no-JS resilience, or an ARIA-compliant accordion pattern with a small JS enhancement if custom open/close animation is required. This should be confirmed with design/UX before build.

3.5 Banner Text (Full-width)

Entry type handle: `bannerText`

Field	Type	Required	Notes
Background Banner	Assets (single)	Yes	
Text	CKEditor (limited config, see 4.1)	Yes	Overlaid on the banner
CTA Label / CTA Link	Shared CTA fields	No	

3.6 Form Component

Entry type handle: `formComponent`

Field	Type	Required	Notes
Form	Wheelform's native field type	Yes	Wheelform ships a field type for exactly this purpose — a dropdown of existing forms configured in the Wheelform control panel section. Add it directly to this entry type's field layout; no custom field needed.

Confirmed installed version: Wheelform 4.0.4 (`xpertbot/craft-wheelform`, Craft 5 compatible).

The Wheelform field type returns a `form` template-service object directly (the same object you'd get from calling `wheelform.form({ id: ... })` manually), so the partial is a thin wrapper around Wheelform's standard render loop:

```
{# _components/formComponent.twig #}

{% macro errorList(errors) %}

  {% if errors %}

    <ul class="errors">

      {% for error in errors %}

        <li>{{ error }}</li>

      {% endfor %}
```

```
        </ul>
    {% endif %}
{% endmacro %}

{% from _self import errorList %}

{% set form = block.formField %}

{{ form.open() }}
    {{ wheelformErrors['form'] is defined ? errorList(wheelformErrors['form']) }}
    {{ wheelformErrors['recaptcha'] is defined ? errorList(wheelformErrors['recaptcha']) }}
    {{ wheelformErrors['honeypot'] is defined ? errorList(wheelformErrors['honeypot']) }}
    {% for field in form.fields %}
        {{ field.render() }}
        {{ wheelformErrors[field.name] is defined ? errorList(wheelformErrors[field.name]) }}
    {% endfor %}
{{ form.close() }}
```

Notes:

- `block.formField` is the entry's Wheelform field handle — rename to match whatever handle is given to the field (e.g. `block.form`).
- `form.open()` / `form.close()` render the `<form>` tag, CSRF input, and hidden fields; keep this outside any `{% cache %}` block, or use Wheelform's `refreshCsrf` option if the page is cached (relevant if Blitz or similar is in use on this site — confirm).
- Submit button styling can be customised via the `submitButton` option if the form is instantiated manually rather than through the field (not needed for the default field-driven flow above, but useful to know if design wants a custom button per instance).
- Field-level markup (checkboxes, radios, selects) can be fully customised by switching on `field.type` instead of calling `field.render()`, if design requires bespoke markup rather than Wheelform's default output — flag with design if that's needed, since it's more build effort than the default loop above.

3.7 4-Column Layout

Entry type handle: `fourColumn`

Field	Type	Required	Notes
Columns	Nested Matrix field, **min blocks = 4** (no max)	Yes	Each column = a mini entry type `column` with **Image** (Assets, required) and **Text** (Plain Text, optional)

Confirmed: reorderable nested Matrix, minimum of 4 columns, no upper limit. When a 5th (or subsequent) column is added it wraps to a new row rather than squeezing into the existing row.

Implementation approach:

- CSS Grid on the front end, e.g. `grid-template-columns: repeat(4, 1fr)` with `flex-wrap`/`grid-auto-rows` so items 5+ naturally start a new row of up to 4.
- On mobile, columns stack to a single column (see 4.2).
- Since there's no max, the template should not assume exactly 4 items when rendering (e.g. avoid hard-coded `nth-child` selectors that only cover 4 columns) — loop generically over `block.columns.all()`.

---

4. Content & Layout Behaviour

4.1 Rich text — limited CKEditor config

Description, Banner Text, and 2-Column text fields all use CKEditor with a **restricted toolbar**, not the default full config. Recommended starting toolbar:

- Bold, Italic
- Link
- Bullet list, numbered list
- Paragraph / Heading (limited to H2–H3 — no H1, to protect page heading hierarchy per 6. Non-Functional Requirements)

No source/HTML view, no tables, no image embedding within these fields (images are handled by their own dedicated Assets fields per component). Exact toolbar to be finalised with content team, but this scope keeps editor output predictable and on-brand.

4.2 Responsive stacking

Confirmed: both 2-Column and 4-Column layouts stack to a single column on mobile. Standard approach:

- CSS Grid/Flexbox with a mobile-first single-column default, expanding to multi-column at the relevant breakpoint (to be confirmed against the site's existing breakpoint tokens — flag to design/dev if none exist yet, so one set of breakpoints is used consistently across all components).
- 4-Column additionally requires the row-wrap behaviour described in 3.7 at the tablet/desktop breakpoint(s).

5. Image Handling

Define named **Image Transforms** per component so editors never have to think about sizing, and so markup can use responsive `srcset` output:

Transform	Used by	Suggested dimensions
`heroBanner`	Header Banner	e.g. 1920×800, quality 82
`fullWidthBanner`	Banner Text	e.g. 1920×600
`contentImage`	1-Col, 2-Col	e.g. 800×600, constrained
`columnThumb`	4-Column	e.g. 400×400

All transforms should generate at least 2x/1x variants for `srcset`, and use lazy-loading (`loading="lazy"`) except above-the-fold instances (Header Banner should NOT lazy-load).

---

6. Editor Experience Notes

- Use clear, non-technical **field instructions** on every field (e.g. "Recommended image size: 1920×800px").
- Group each entry type's fields under a single tab named after the component for clarity in the Matrix UI.

- Set sensible **entry type icons/colours** in the CP if the Craft version/plugins in use support it, to make the component picker scannable.

---

7. Non-Functional Requirements

- **Accessibility:** semantic heading levels per component (confirm heading hierarchy rules with design so Title/Text fields map to correct `<h1>`–`<h3>` per page, not per component in isolation); accordion must be keyboard-operable; limited CKEditor toolbar (4.1) excludes H1 to protect this.
- **Performance:** all images served via Craft transforms with responsive `srcset`; avoid rendering empty markup for optional fields (CTA, Description) rather than emitting empty elements.
- **SEO:** Header Banner Title should be usable as (or map to) the page's primary heading where the component sits at the top of the page — confirm with SEO/content strategy so it isn't duplicated with a separate SEO title field.

---

8. Resolved Decisions Log

All initial open questions have been answered:

#	Question	Decision
1	Wheelform version / render API	**Wheelform 4.0.4** — confirmed API documented in Section 3.6
2	2-Column variants beyond the three named	**No new variants** — but an optional CTA button has been added to the component (Section 3.3)
3	4-Column: fixed grid vs. reorderable Matrix	**Reorderable nested Matrix**, min 4 columns, no max; 5th+ column wraps to a new row, stacks on mobile (Section 3.7)
4	Rich text scope for Description/Banner Text	**Limited CKEditor config** — restricted toolbar, no H1 (Section 4.1)

#	Question	Decision
5	Breakpoints for column stacking	**Stacks to single column on mobile** for both 2-Column and 4-Column; exact breakpoint values to confirm against existing design tokens (Section 4.2)

Remaining open item: confirm the site's existing breakpoint tokens (or agree new ones) so 2-Column, 4-Column, and the 4-Column row-wrap behaviour all use the same values — flag to design/dev before front-end build starts. - **Andre to decide best breakpoints approach**