User Blueprints + Blueprint Adapter + Global Inspector Registry — Specification (v1, Draft)

Status: *Draft specification.*
Related:

- `docs/future/user-blueprints-and-global-inspector.requirements.v1.md` *(requirements)*
- `docs/future/user-blueprints-and-global-inspector.conversation-notes.v1.md` *(rationale)*
- `src/data/card-templates.ts` *(template metadata + legacy component mapping)*
- `src/data/blueprints.ts` *(built-in blueprints)*
- `src/types/blueprints.ts` *(blueprint schema)*
- `src/components/BlueprintRenderer/index.tsx` *(current renderer)*
- `src/data/inspector-fields.ts` + `src/types/inspector.ts` *(current inspector mapping)*
- `src/types/card-data.ts` / `src/types/cards-db.ts` *(card data + DB records)*
- `src/lib/card-record-mapper.ts` *(record ⇄ data mapping)*

---

1. Overview

This spec defines how to evolve the current **template + blueprint** system to support:

- **User-defined blueprints** (persisted layout definitions)
- A **runtime blueprint adapter** for controlled layout augmentation
- A **global inspector field registry** with blueprint-driven inclusion

The goal is to keep rendering deterministic and template-driven while allowing a **generic/composable blueprint** workflow for advanced users. This spec focuses on the architectural transition, not low-level component implementation details.

3.3 Renderer

- `BlueprintRenderer` (`src/components/BlueprintRenderer/index.tsx`) resolves:
- `background`, `border`, `image`, `text`, `title` layers
- a bottom-anchored `stack` group for dynamic content
- Group children support `text`, `stats-hero`, `stats-monster`, `icon` (rendered via `GroupIconLayer`).

3.4 Inspector

- Two modes exist today:
- **Legacy** per-template inspector forms (e.g. `MonsterInspectorForm`, `HeroInspectorForm`).
- **Generic** inspector driven by `inspectorFieldsByTemplate` (`src/data/inspector-fields.ts`).
- Field types are defined in `src/types/inspector.ts`.

3.5 Card Data & Persistence

- Card data shape is in `src/types/card-data.ts` (template-specific data objects).
- Persisted records are in `src/types/cards-db.ts` with flattened fields.
- Mapping between DB and editor data is in `src/lib/card-record-mapper.ts`.
- Drafts and active cards are stored in `localStorage` via `CardEditorContext`.

---

4. Target Architecture (High Level)

The system will support two parallel sources of layout:

- **Built-in templates**: continue to use `templateId` → built-in blueprint mapping.
- **User blueprints**: persisted layouts referenced by `blueprintId` in card records.

A **blueprint resolution step** will run before rendering, allowing controlled modifications based on card data and configuration. Inspector rendering will be driven by a **global field registry**, using blueprint binds and card data presence to decide what to show.

---

6. Blueprint Adapter (Resolution Step)

6.1 Purpose

The adapter is a deterministic transform that:

- Takes a **base blueprint** (built-in or user blueprint),
- Applies **configuration rules** derived from card data,
- Outputs a **resolved blueprint** used for rendering.

6.2 Determinism & Purity

- The adapter MUST be deterministic:
- Same inputs → same output.
- The adapter MUST NOT mutate the input blueprint object.
- The adapter MUST preserve the persistent blueprint identity.

6.3 Allowed Transformations

The adapter MAY:

- Insert optional layers or group children that are defined as “slots” in the base blueprint.
- Move elements between top-level layers and groups.
- Reorder layers or group children where explicitly allowed.
- Adjust bounds/origins if declared as mutable by the base blueprint.

The adapter MUST NOT:

- Reliably infer insertion points from heuristics (e.g. “first stack group”).
- Break deterministic export/preview parity.

6.4 Insertion Points / Slots

Composable blueprints MUST expose explicit insertion points:

- Stable layer or group IDs where optional components can be inserted.
- This avoids fragile assumptions and makes the adapter predictable.

8.2 Global Field Order

A single global order MUST be defined and used for inspector rendering.

8.3 Inclusion Rule (Blueprint + Card Data)

A field MUST be included in the inspector only if:

1. The resolved blueprint binds to the primary key, AND
2. The card data model includes the primary field.

Secondary controls MUST only appear when the primary field is included (e.g. `titleStyle` only when `title` is present and bound).

8.4 Blueprint Bind Introspection

The system MUST derive "used field keys" by scanning:

- `layers[*].bind` (e.g., `textKey`, `imageKey`, `titleKey`)
- `groups[*].children[*].bind` (e.g., `iconKey`)

This MUST work for built-in and user blueprints.

---

9. User Blueprint UX

9.1 Creating a card with a user blueprint

- The UI MUST provide a path to start from a generic/composable blueprint.
- Selecting this path MUST create (or assign) a persisted user blueprint record.

9.2 Saving and sharing

- If a blueprint is referenced by a single card, edits MUST save in place.
- If a blueprint is referenced by multiple cards, edits MUST fork into a new blueprint.
- The UI MUST clearly communicate when an edit will fork.

9.3 Save as template (user blueprint reuse)

- Users MUST be able to save a blueprint as a reusable template by providing `name` and `kind`.
- The saved blueprint MUST be available as a selectable option for other cards.

12. Constraints & Defaults

- Max counts (icons per row, rows per card) MUST be enforced at the UI/business-logic layer.
- Layout defaults (e.g. `monsterIconLayout = "start"`) MUST be specified and applied when missing.
- Rendering MUST be deterministic; no layout randomness.

---

13. Acceptance Criteria

- Cards can reference either `templateId` or `blueprintId` (mutually exclusive) and render correctly.
- A user blueprint referenced by one card can be edited in place; edits reflect in preview/export.
- A user blueprint referenced by multiple cards cannot be edited in place; edits fork.
- Inspector fields are derived from a global registry and only show when both blueprint + data include the field.
- Missing assets are visible in SVG and highlighted in the inspector.
- Export/import round-trips preserve user blueprints and card references.

---

14. Deferred Items

- Stable IDs for repeatable rows/icons (required for robust selection/reorder).
- Dedupe strategy for identical user blueprints.
- Title bottom behavior (overlay vs consume space) for the generic/composable blueprint.