User Blueprints + Blueprint Adapter + Global Inspector Registry — Requirements (v1)

Status: *Requirements only (no implementation details).*
Related: `docs/future/user-blueprints-and-global-inspector.conversation-notes.v1.md` *(full rationale/background),* `docs/future/database-export-import.requirements.md` *(export/import baseline),* `src/components/BlueprintRenderer/index.tsx` *(current renderer),* `src/data/blueprints.ts` *(built-in blueprints),* `src/data/inspector-fields.ts` *(current per-template inspector mapping).*

1. Overview

Add support for **user-defined card layouts** by allowing users to create, persist, and reuse **blueprints** (render plans). At the same time, simplify inspector authoring by replacing per-template inspector field lists with a **global Field Registry** and **global ordering**, where inclusion is derived from blueprint binds and card data presence.

The system remains **template-driven** and **deterministic** (preview/export consistency). This feature introduces a **generic/composable blueprint** path for advanced customization, without requiring a fully general drag/drop canvas editor in v1.

2. Goals & Non-Goals

2.1 Goals

- Users can create cards that reference a **persisted user blueprint** (not just hard-coded templates).
- Users can reuse a saved blueprint across multiple cards (a “custom template” in user-facing terms).
- The inspector UI is driven by:
- a **global field registry** (UI semantics per field key),
- a **global field order**,
- and **blueprint-driven inclusion** (based on binds) with card-data gating.
- Rendering remains deterministic:

4.2 User blueprint records

- The system MUST persist user blueprints in a dedicated store/table/collection.
- Each user blueprint record MUST have:
- a persistent unique identifier (`blueprintId`),
- a `schemaVersion` (initially `1`),
- the blueprint definition (layers/groups/binds/props/bounds as required by the renderer),
- user-facing metadata for template selection (at minimum: `name` and `kind`/category).
- The system SHOULD store timestamps (e.g. `createdAt`, `updatedAt`) for user blueprints.

4.3 Sharing and mutability rules

- A user blueprint referenced by exactly one card MUST be editable in-place (no forced “unlock” step).
- A user blueprint referenced by more than one card MUST be treated as **locked**:
- Editing MUST NOT mutate the shared blueprint in-place.
- Editing MUST produce a new blueprint record (“fork”) and update only the current card to reference it.
- The UI MUST clearly communicate when an edit will create a new blueprint (“Save as new blueprint” / “Fork”).

4.4 Repeatables storage (embedded arrays)

- The system MUST support repeatable component data using embedded arrays within card data, not relational join tables.
- The system MUST support backward-compatible reading where feasible (e.g. legacy single-value fields may be interpreted as the first array item where appropriate).
- Any migration strategy MUST be safe against interruption (e.g. refresh) and MUST avoid leaving storage in an invalid state.

5. Blueprint Adapter (Runtime Augmentation) Requirements

5.1 Purpose

- The system MUST support resolving a base blueprint into a rendered blueprint via a deterministic adapter step.
- The adapter MUST enable controlled structural changes such as:
- moving an element between top-level layers and a group,
- reordering layers,

7.1 Global Field Registry

- The system MUST define a single global Field Registry keyed by field key (e.g. `title`, `description`, `imageAssetId`).
- For each field key, the registry MUST define:
- UI control type (text, textarea, asset picker, composite editor, etc.),
- label source (i18n key or equivalent),
- validation/constraints where applicable.

7.2 Global field order

- The system MUST define a single global field order for inspector rendering.
- v1 MUST NOT require per-template reordering support.

7.3 Inclusion rules (blueprint + card data gating)

- A field/editor MUST be shown in the inspector only if BOTH are true:
- The resolved blueprint binds/uses the relevant primary field key, AND
- The card data model contains the relevant primary field (or section) for the card.
- Secondary option controls (e.g. style/layout controls) MUST be shown only when their primary bound field is included.

7.4 Blueprint introspection

- The system MUST be able to derive “used field keys” from a blueprint by scanning layer/group binds.
- This derivation MUST work for both built-in and user blueprints.

8. User Experience Requirements

8.1 Creating and using user blueprints

- The UI MUST provide a way to:
- create a card using a generic/composable blueprint,
- save changes to that blueprint when it is unshared,
- save a blueprint as a reusable custom template (name + kind),
- choose an existing user blueprint for a new card.

8.2 Locked/shared blueprint edits

- When a user attempts to edit a locked/shared blueprint, the UI MUST:
- prevent in-place mutation, AND
- offer a clear “save as new blueprint” (fork) path.

- Title positioning semantics for generic/composable blueprints (overlay vs consuming layout space).