THE BRIDGE: Full Technical Specification (3P3 Methodology)

Author: Osbert Vulor (Caufero)

Purpose: Technical Document for developing THE BRIDGE.

---

1. Introduction

Purpose of this document:

Lay out the complete technical plan for THE BRIDGE: philosophy, architecture, database design, JSON templates, rules engine, UI, security, integrations, reporting, testing, deployment, and maintenance. This is the working spec for development and future handover.

Audience:

Developers, architects and technical stakeholders.

Background:

THE BRIDGE is built on Luca's 3P3 direction: everything is a process. The approach will have the system built on three tables and a small set of generic engines, so the app is metadata-driven instead of hard-coded.

2. Vision & Philosophy

Everything is One. Everything is a Process:

We will not be shipping many separate modules. We ship one engine that can express many modules. The user defines a process; the system renders forms, runs steps, logs actions, and stores results.

Materials → Action → Result:

Every operation binds three things: the object we work on (material), the action we take, and the outcome. We store this consistently so we can slice history from any angle.

1P = 1E = 1C. Each process instance carries its own entity and composition context. Practically, it means every run has identity, data, and relationships packaged together through JSON. The code reads one definition and knows what to render, who to involve, and what to persist.

Keep Schema Simple:

We will keep the schema simple and small. We avoid heavy calculated fields and table relationships.

3. System Objectives

- **Users will define their own process templates:** A Process Template Designer will enable admin users to create new “modules” by creating templates (no schema changes).
- **Centralized logging:** One ledger (Logs) records every instance and step.
- **Speed and scale:** Lean layouts, indexed fields, ExecuteSQL for targeted reads.
- **Transparency:** Clear task assignment, status, timestamps, and actor history.

---

4. Functional Requirements

- Create and publish **process templates** (attributes, steps, roles, rules) via a UI.
- Start, view, edit, and advance **process instances**.
- Manage **entities** (staff, clients, products, etc.) as processes.
- **Approvals** and **Denials** with notes.
- **Daily reporting** and dashboards from the central ledger.
- **Notifications** (Email) via template actions.
- **Authentication & roles**; field-level visibility and validation.

5. Non-Functional Requirements

- **Performance:** Responsive UI for users
- **Security:** Role-based access, field masking where required, audit of sensitive actions.
- **Reliability:** Server-hosted, with daily backups and weekly tested restores.
- **Maintainability:** JSON templates, small schema, clear scripts, documented rules.
- **Availability:** Target business-hours availability with scheduled maintenance windows.

---

6. System Architecture

Schema: Three tables. No relationships.

Files: One main file.

Stack: FileMaker + HTML/JavaScript for dynamic forms; ExecuteSQL for targeted reads; server-side scripts for scheduled jobs.

Integrations: Email integrations and other possible integrations that might come up.

---

7. Database Design (3-table, JSON-driven)

7.1 Processes (template definitions)

Purpose:

Holds one row per template version. The JSON here tells the app how to render and route.

Template JSON spec (high-level):


```
{
  "name": "Create Product",
  "objectType": "PRODUCT",
  "fields": [
    {"code":"sku","label":"SKU","type":"text","required":true,
     "unique":true,
     "writeTo":{"table":"PRODUCTS","field":"SKU"}},
    {"code":"name","label":"Name","type":"text","required":true},
    {"code":"price","label":"Unit Price","type":"number","min":0},
    {"code":"active","label":"Active","type":"boolean","default":true}
  ],
  "steps": [
    {"code":"draft","label":"Draft","role":"SalesOps"},
    {"code":"review","label":"Review","role":"ProductMgr"},
    {"code":"approve","label":"Approve","role":"Director",
     "onApprove":[{"action":"writeThrough"}]}
```

```
  ],
  "rules": {
    "visibility": [],
    "validations": [],
    "notifications": [{"event":"approve","type":"slack","channel":"#ops"}]
  }
}
```


7.2 Logs

Purpose:

One row per instance/step chain; stores values for the run and who did what, when.

7.3 Entities (ETY)

Purpose:

One table that store the components of entities.

8. Data Flow

1. **Define template:** Admin creates a row in **Processes** with TemplateJSON.
2. **Start instance:** User starts a run → app reads template, renders form, creates **ProcessLogs** row.
3. **Capture values:** Form submissions update ValuesJSON in **ProcessLogs**.
4. **Advance steps:** Engine references steps to change CurrentStep, Status, and AssignedTo.
5. **Actions:** Rules trigger notifications, write-through to **Entities** or first-class tables, or spawn child logs.
6. **Report:** Operational queries hit **Logs**; Analytics may read **Entities (ETY)** table also.

DNA code generation:

We use type+year+sequence (e.g., TSK25001). A small sequence table prevents collisions.

Lifecycle:

Old instances keep their original template version; new instances use the latest.

9. Core Modules

The Processes Template Designer (heart of the system).

- Create/edit **Templates** (name, code, version, definition, objectType).
- Add **Fields** (code, label, type, required, options, visibility, validation, write-through mapping).
- Configure **Steps** (code, role, order, auto-advance, completion rules).
- Add **Actions** (notifications, write-through, spawn processes, auto-numbering).
- Validate and **Publish** the template.

---

10. User Interface Design

- **Renderer (generic form):** One layout renders controls from TemplateJSON. Visibility and validation come from rules.
- **Menu:** Process templates are listed on the menu. So if a new process template is created and approved, it appears on the menu.

UI rules:

- Keep layouts light; avoid heavy portals.
- Use scripted finds and ExecuteSQL for lists.
- Cache template JSON during a session when safe.

---

11. Security Model

- **Authentication**: Standard FileMaker accounts
- **Audit**: Every action is stored in the 3 tables:
  - Processes
  - ETY
  - Logs

---

12. Rules Engine (For Processes)

Rule types (based on template definition):

- **Visibility** – hide/disable fields based on conditions.
- **Validation** – required, unique, regex, min/max, cross-field logic.
- **Actions** – notifications, write-through, spawn child process, auto-codes.

Execution:

Renderer loads rules from JSON, evaluates in order, raises messages, and prevents save if invalid. Server-side re-validation on commit for safety.

---

13. Integration Strategy

- **Email**: Actions in rules can notify people or departments on events.
- **Webhooks**: Template actions can call external services if needed later.

---

14. Reporting & Analytics

- **Operational** – lists and dashboards driven by **Processes, ETY and Logs**
- **Saved reports** – reports are generated on the fly when an event is triggered
- **Exports** – CSV/XLSX/PDF as needed.

15. Development Plan

Duration:

With this lean approach, will do well to do core development within 3 - 4 months.

Environment:

Build directly on Luca's server.

Phases:

1) **Core Engine Development:** Process Template builder, versioning, validations, write-through.
2) **Seed templates:** Products, Leave Approval, Add Staff, Approve User.
3) **Dashboards & reports.** Operational views; daily reporting.
4) **Security & polish.** Field visibility, performance checks, admin tools.

16. Testing Strategy

- **Unit tests**: Render, save, advance, rules evaluation.
- **Template tests**: Each seed template gets fixtures (sample JSON + sample ValuesJSON).
- **Integration tests**: Rrite-through into Entities/first-class tables.
- **UAT**: run a real process end-to-end with the team; measure time and clarity.
- **Bug tracking**: one list with severity and repro steps; short cycles.

---

17. Deployment Plan

- **Server setup**: Luca's FileMaker Server
- **Files**: TheBridge.fmp12.
- **Backups**: nightly; keep backups for 14–30 days.

---

18. Documentation

- **Developer docs**: JSON spec, renderer contract, rules reference, script catalog.
- **User guide**: starting a process, working My Work, submitting/approving.
- **Admin guide**: building templates, managing roles, versioning.
- **Knowledge transfer**: short videos for the core flows.

19. Maintenance & Support

- **Support process**: Ticket intake, triage, fix, release.
- **Template changes**: New version → existing instances keep old; migration only when safe.
- **Scalability**: Add indexes as data grows; materialize tables only where justified by reporting needs.

This is our build spec. It reflects the 3P3 philosophy and gives us a practical, lean path to deliver quickly and scale cleanly. Once we agree on the open points, we start development.