**GenWizard — Architecture Generator and Pattern Builder**

Owner: Siva Teja Narayana

Date: 04 Sep 2025

# Executive Summary

GenWizard is an AI-driven automation platform used to generate architecture assets and reusable execution patterns. Two flagship capabilities are:

* Architecture Generator — delivered in v2.0 (GenAI-first) and v4.0 (Agentic AI).
* Pattern Builder — a marketplace-backed engine for configurable, reusable step-flows ('patterns') that produce stories, designs, architecture diagrams, OpenAPI specs, code scaffolds, and tests.

Architecture Generator: converts sparse inputs (BRD/NFR, repos, PDFs, legacy diagrams) into a Suggestive Content table → architecture diagrams → narrative → review checklist. v4.0 raises first-pass quality via multi-agent verification and targeted repair loops.

Pattern Builder: executes sequences of steps (prompt/API/data/JSON/diagram) with Decision Gates (LLM+RAG) after each step to proceed or re-execute. Approved patterns can be published to the Marketplace.

# Table of Contents

* 1) Module A — Architecture Generator
* 1.1 Goal · Inputs · Outputs
* 1.2 Phase 1 — v2.0 (GenAI-first) + LLM Integration
* 1.3 Phase 2 — v4.0 (Agentic AI) + LLM Integration
* 1.4 Reverse vs. Modernize (side-by-side)
* 1.5 Pitfalls & Mitigations
* 1.6 Policies · Security · Governance
* **Sample Prompt**
* 2) Module B — Pattern Builder (v2.0)
* 2.1 Goal · Inputs · Outputs
* 2.2 Overview (Mermaid)
* 2.3 Step Library
* 2.4 LLM Integration (Decision Gates, RAG, Prompts)
* 2.5 Sample End-to-End Patterns
* 2.6 Example Pattern Definition (YAML)
* 2.7 Marketplace Flow · Governance · Ops
* Appendix — Prompt Snippets (quick copy/paste)

# Module A — Architecture Generator

## 1.1 Goal · Inputs · Outputs

Goal: Turn minimal context (brief/BRD/NFR/US/Features/Design Docs/ repos, or a legacy diagram) into review-ready architecture diagrams and a short narrative across Functional, Application/Solution, and Technical/Infra views. Flow: Suggestive Content table → user confirmation → final diagram + narrative → review checklist.

### Inputs

* Mode: Greenfield, Reverse Architecture, Modernize Architecture
* Artifacts (optional): BRD/NFR PDFs, ADRs/tickets, read-only code repos, legacy diagram (PNG/SVG)
* Constraints: cloud/vendor stance, security posture, data residency, latency/throughput targets, observability baseline, naming rules

### Outputs

* Suggestive Content table (assumptions, open questions, layers, components, key edges)
* Architecture diagram assets (PNG/SVG) + short narrative (choices, trade-offs, NFR coverage, risks)
* Review checklist + validation notes (rules/geometry checks)
* Telemetry (per-phase tokens/latency), export checksums

## 1.2 Phase 1 — v2.0 (GenAI-first)

Sequential LLM calls: short questionnaire → drafts → architecture diagram → export, with optional inline edit. Reverse uses RAG over code; Modernize uses a vision model over a legacy image.

v2.0 overview :

A diagram of a diagram

AI-generated content may be incorrect.

### LLM Integration — v2.0 (all details)

|  |
| --- |
| **Call sites (sequence)**  1) Questionnaire 2) Functional draft 3) Application draft 4) Technical primitives  5) Reverse (RAG) 6) Modernize (Vision) 7) Diagram generation 8) Geometry repair (optional) |

|  |
| --- |
| **P1 — Questionnaire**  Role: Elicitor  Goal: Ask only what changes the diagram; MCQs  Inputs: diagram\_type, brief, NFRs, artifacts?  Output: [ { id, text, options[], why } ]  Notes: 6–10 Qs; plain English |

|  |
| --- |
| **P2 — Functional draft**  Role: Functional modeler  Goal: 6–8 layers; 8–11 components/layer; inter-layer edges only  Inputs: brief, answers\_from\_P1  Output: { layers[], components[], edges[], rationale{} }  Notes: short alphanumeric names |

|  |
| --- |
| **P3 — Application draft**  Role: App architect  Goal: Presentation → API Gateway → Services → Data → Observability → Security (+ nfr\_checks)  Inputs: design summary, answers\_from\_P1  Output: { layers[], components[], edges[], nfr\_checks[] } |

|  |
| --- |
| **P4 — Technical primitives**  Role: Platform planner  Goal: user→edge→compute→data→observability→externals  Inputs: NFRs, integrations, security constraints  Output: { layers[], components[], edges[], layout\_hints{ max\_per\_row:5, layer\_gap:40 } } |

|  |
| --- |
| **P5 — Reverse (RAG)**  Role: Code cartographer  Goal: Boundaries, flows, risks  Inputs: repo summary (endpoints, deps, schemas), change signals  Output: { layers[], components[], edges[], risks[] }  Notes: cite 1–2 file paths/lines |

|  |
| --- |
| **P6 — Modernize (Vision)**  Role: Renovation guide  Goal: Modern target + migration notes  Inputs: PNG/SVG legacy diagram, constraints  Output: { layers[], components[], edges[], modernization\_notes[] } |

|  |
| --- |
| **P7 — Diagram generation**  Role: Diagram producer  Goal: Layered architecture diagram + 6–8 bullet narrative  Inputs: spec from P2/P3/P4/P5/P6  Output: diagram asset + narrative  Notes: ≤5 items/row; ≥40px between layers; wrap text |

|  |
| --- |
| **P8 — Geometry repair**  Role: Layout fixer  Goal: Remove overlaps/crossings; keep semantics  Inputs: current diagram, issues  Output: corrected diagram |

|  |
| --- |
| **Grounding & retrieval (Reverse)**  Build a light index from code (APIs, dependencies, schemas, change signals).  Fetch 6–10 snippets with source+freshness; dedupe by section; recency boost. |

|  |
| --- |
| **Validation & repair**  Layer-only edges; alphanumeric names; ≤5 components/row; ≥40px between layers.  Geometry checks → if fail → targeted P8 repair.  Retrieval hygiene: recency scoring, pinned sources, section-level dedupe. |

|  |
| --- |
| **Ops & safety**  Token/latency budgets per step; output schema checks; idempotent exports; checksums.  Redacted logs, audit IDs; Keycloak-guarded proxy. |

## 1.3 Phase 2 — v4.0 (Agentic AI)

A stateful multi-agent graph replaces the linear flow with specialization, verification, and targeted repair loops to improve first-pass quality and reduce manual edits.

v4.0 overview:

### A diagram of a company AI-generated content may be incorrect.

### LLM Integration — v4.0 (all details)

|  |
| --- |
| **Intake**  Normalize inputs; classify mode; seed facts\_ledger. |

|  |
| --- |
| **Planner**  Phases, exit criteria, budgets; Output: plan.json. |

|  |
| --- |
| **Questionnaire**  ≤4 MCQs/round; updates facts\_ledger.json (diffed). |

|  |
| --- |
| **Discovery/RAG**  6–10 cites with source+freshness; hybrid retrieval; dedupe by section. |

|  |
| --- |
| **FAD / AAD / TAD**  Functional/Application/Technical drafts with rationale + citations; layer-only edges. |

|  |
| --- |
| **Vision-Modernizer**  Read image; propose modern target + phased migration notes. |

|  |
| --- |
| **Repo-Reverse**  Infer boundaries/ports/data stores; list anti-patterns/risks. |

|  |
| --- |
| **Integration Mapper**  External systems + edge notes (auth, retry/timeout). |

|  |
| --- |
| **Diagram Synthesizer**  Layered architecture diagram (≤5/row; ≥40px gaps; wrap text). |

|  |
| --- |
| **Geometry Optimizer**  Spacing, overlaps, crossings; preserve IDs. |

|  |
| --- |
| **Validator**  Schema/rules/XML checks; emit fix-plan (who fixes, how). |

|  |
| --- |
| **Reviewer (HITL)**  Approve / targeted Repair / Defer; ≤10 bullets summary. |

|  |
| --- |
| **Exporter**  PNG/SVG + checksums; idempotent keys. |

|  |
| --- |
| **Telemetry/Cost**  Per-agent spans; token/latency budgets; anomaly alerts. |

|  |
| --- |
| **Risk & Policy**  PII redaction; deny-lists; policy logs. |

|  |
| --- |
| **Coordination & state**  Shared artifacts: plan.json, facts\_ledger.json, cites.json, latest\_spec, latest\_diagram.  Router obeys budgets and routes repair to the single failing agent only. |

|  |
| --- |
| **Validation & repair**  Validator pinpoints the failed rule + agent + minimal variables.  Auto-repair loops are scoped (fast/cheap) and keep prior success intact. |

|  |
| --- |
| **Ops & safety**  OpenTelemetry spans per agent; token/latency budgets; anomaly flags.  Pinned renderer; idempotent exports; checksums; redacted logs; Keycloak; project-scoped indices. |

## 1.4 Reverse vs. Modernize (side-by-side)

* Reverse — v2.0: RAG → infer boundaries/risks → architecture diagram → export.
* Reverse — v4.0: Repo-Reverse → AAD → Diagram → Geometry → Validate → Reviewer → Export (citations preserved; targeted repair loops).
* Modernize — v2.0: Vision extract → modernization patterns (strangler, event bus, API GW, observability baseline) → diagram → export.
* Modernize — v4.0: Vision-Modernizer → AAD/TAD → Integration Mapper → Diagram → Geometry → Validate → Reviewer → Export (adds phased migration notes).

## 1.5 Pitfalls & Mitigations

* Over-eager modernization → keep suggestions silent unless requested or clearly warranted.

## 1.6 Policies · Security · Governance

* Diagram conventions: layer-only edges; ≤5 components/row; ≥40px vertical gap; short alphanumeric names; wrap text.
* Security: Keycloak-secured proxy; project ACLs; image metadata scrubbed; audit IDs; redacted logs.
* Ops: per-phase budgets; idempotent exports; checksums; retriable steps; OpenTelemetry spans.

**Sample Orchestrator Prompt:**

# Module B — Pattern Builder (v2.0)

## 2.1 Goal · Inputs · Outputs

Goal: Let teams author reusable workflows ('patterns')—each a sequence of steps (prompt, API, data aggregation, JSON transform, diagram generation)—that reliably produce delivery artifacts. After each step, a Decision Gate (LLM+RAG+previous output) decides proceed or re-execute with a focused repair prompt. When the creator marks the run 'Satisfied', publish to the Marketplace.

### Inputs

* Pattern intent (e.g., Feature → Stories → Design Doc → Architecture diagram → API Spec → Code Skeleton → Tests)
* Artifacts: brief/BRD/NFR, repo snippets, tickets, standards (optional)
* Constraints: style guides, naming, cloud/vendor stance, data residency, coverage targets
* Runtime: active steps, per-step LLM/provider, temperature, max tokens, budgets

### Outputs

* Run artifacts (markdown/docs, specs, code scaffolds, tests, architecture diagram)
* Decision reports (why proceed vs re-execute, citations, diffs)
* Pattern definition (YAML/JSON) ready for the Marketplace

## 2.2 Overview

A diagram of a process

AI-generated content may be incorrect.

## 2.3 Step Library

* Prompt Step — stories, design doc, architecture diagram, OpenAPI, code scaffold, tests.
* API Step — call tools (linters, validators, test runners, CI hooks).
* Data Aggregation Step — RAG over BRD/NFR, code, issues; merge snippets.
* JSON Transform Step — normalize/remap JSON for downstream tools.
* Diagram Generation Step — produce an architecture diagram asset.
* Decision Gate (built-in) — evaluate quality and either proceed or re-execute with a targeted fix.

## 2.4 LLM Integration (Decision Gates, RAG, Prompts)

|  |
| --- |
| **Where models run**  1) Prompt Steps — stories, design doc, architecture diagram, API spec, code scaffold, tests.  2) RAG Enrichment — gather 6–10 passages with source + freshness for the next step.  3) JSON Transform — compact JSON (optional).  4) Decision Gate — LLM + RAG checks the artifact and returns proceed vs re-execute + one focused repair prompt. |

|  |
| --- |
| **Decision Gate specification**  Inputs: previous artifact, cites.json (RAG), validator/CI reports (if any).  Output: { decision: proceed|reexecute, reason, targeted\_prompt\_if\_reexecute }.  Constraints: reason ≤2 bullets; repair prompt ≤80 words; fix only the top 1–2 gaps. |

|  |
| --- |
| **Stories (Prompt)**  Role: Agile writer  Goal: INVEST stories w/ 3–5 ACs each  Inputs: feature brief, personas, constraints  Output: stories.md (≤12)  Notes: label epics vs stories |

|  |
| --- |
| **Design Doc (Prompt)**  Role: System designer  Goal: scope, flows, data, trade-offs, risks, open questions  Inputs: stories.md, constraints  Output: design.md (≤2 pages) |

|  |
| --- |
| **RAG Enrichment (Data Aggregation)**  Role: Research assistant  Goal: 6–10 snippets with source + freshness  Inputs: design.md, doc index  Output: cites.json {text, source, offset, freshness} |

|  |
| --- |
| **Architecture Diagram (Prompt)**  Role: Solution architect  Goal: layered architecture diagram + ≤8-bullet narrative  Inputs: design.md, cites.json, chosen style  Output: diagram asset(s) + narrative  Notes: alphanumeric names; edges between layers |

|  |
| --- |
| **API Spec (Prompt)**  Role: API designer  Goal: minimal, valid OpenAPI (auth, core paths, schemas)  Inputs: stories.md, design.md, cites.json  Output: openapi.yaml |

|  |
| --- |
| **Code Scaffold (Prompt)**  Role: Scaffolder  Goal: folder layout + 1–2 endpoint stubs  Inputs: openapi.yaml, lang/framework prefs  Output: scaffold.txt + starter code |

|  |
| --- |
| **Tests (Prompt)**  Role: Test author  Goal: unit + integration tests mapped to ACs  Inputs: stories.md, openapi.yaml  Output: tests.md (+ optional snippets) |

|  |
| --- |
| **JSON Transform (Transform)**  Role: Transformer  Goal: compact normalized.json  Inputs: openapi.yaml or JSON/YAML  Output: normalized.json |

|  |
| --- |
| **Validator/CI (API Step)**  Role: External tool invoker  Goal: lint/validate or run tests  Inputs: openapi.yaml, tests.md  Output: validator\_report.json |

|  |
| --- |
| **Grounding & RAG**  Sources: BRD/NFR, code, tickets, standards; section-level dedupe; recency-weighted ordering.  Gates must cite ≥1 snippet when rejecting. |

|  |
| --- |
| **Ops & safety**  Per-step token/latency budgets; retries/circuit breakers; run telemetry + diffs at every gate.  Keycloak-guarded APIs; project-scoped indices; redacted logs + audit IDs; allow/deny lists; PII redaction. |

## 2.5 Sample End-to-End Patterns

Pattern A — Feature → Release

1. User Stories → Design → RAG → Architecture diagram → OpenAPI → Scaffold → Tests → Validator/CI
2. Gate after each step; publish when creator marks Satisfied and latest run is all 'proceed' with CI green.

Pattern B — Story-first Quality

1. Stories → Gate (reject non-INVEST) → Design → Gate (flows/data/risks) → Tests → Gate (coverage & edge cases) → publish

Pattern C — Spec-to-SDK

1. OpenAPI (Prompt or Import) → JSON Transform (normalize) → API Step (SDK generator) → Gate (lint & compile) → publish

## 2.6 Example Pattern Definition (YAML)

name: Feature\_to\_Release  
version: "1.0"  
steps:  
 - id: stories  
 type: prompt  
 model: gpt-4o  
 prompt\_ref: user\_stories  
 - id: design  
 type: prompt  
 model: gpt-4o  
 prompt\_ref: design\_doc  
 - id: cites  
 type: data\_aggregation  
 strategy: rag  
 sources: [brd, nfr, repo]  
 - id: diagram  
 type: prompt  
 model: gpt-4o  
 prompt\_ref: architecture\_diagram  
 - id: api  
 type: prompt  
 model: gpt-4o  
 prompt\_ref: openapi\_minimal  
 - id: scaffold  
 type: prompt  
 model: gpt-4o  
 prompt\_ref: code\_scaffold  
 - id: tests  
 type: prompt  
 model: gpt-4o  
 prompt\_ref: tests\_from\_ac  
 - id: validator  
 type: api  
 service: openapi\_lint  
gates:  
 - after: stories  
 type: decision  
 prompt\_ref: gate\_quality  
 - after: design  
 type: decision  
 prompt\_ref: gate\_quality  
 - after: diagram  
 type: decision  
 prompt\_ref: gate\_quality  
 - after: api  
 type: decision  
 prompt\_ref: gate\_quality  
 - after: tests  
 type: decision  
 prompt\_ref: gate\_quality  
publish:  
 require\_creator\_approval: true  
 require\_last\_run\_all\_proceed: true  
 attach\_artifacts:  
 - stories.md  
 - design.md  
 - diagram.png  
 - openapi.yaml  
 - tests.md

## 2.7 Marketplace Flow · Governance · Ops

1. Creator review → mark Satisfied
2. Eligibility → latest run all 'proceed'; CI/lint green
3. Metadata → name, tags, inputs/outputs, languages, license
4. Publish → pattern becomes discoverable & cloneable
5. Versioning → immutable runs; updates = new versions with changelog

* Auth: Keycloak-protected endpoints; per-team pattern ACLs
* Data: project-scoped indices; sensitive fields redacted; audit IDs
* Safety: Decision Gates enforce allow/deny lists, PII redaction, citation requirements
* Ops: per-step token/latency budgets; retries/circuit breakers; telemetry + diffs at each gate

# Appendix — Prompt Snippets (quick copy/paste)

|  |
| --- |
| **gate\_quality**  Check the artifact against the stories/design/NFRs. If OK: decision=proceed (≤2 bullets).  Else: decision=reexecute with one focused repair prompt (≤80 words) for the top 1–2 gaps. |

|  |
| --- |
| **architecture\_diagram**  Generate a layered architecture diagram (UI/API/Services/Data/Observability/Security).  Use alphanumeric names and layer-to-layer edges. Provide ≤8 bullets explaining key choices. |

|  |
| --- |
| **tests\_from\_ac**  Write test cases mapped to each user story’s ACs. Include happy path + edge cases. Keep concise and runnable. |

|  |
| --- |
| **openapi\_minimal**  Draft a minimal OpenAPI (auth, main paths, request/response, error schema). Keep valid and short; mark TBDs. |

|  |
| --- |
| **code\_scaffold**  Produce a project scaffold for {lang}/{framework} with folders, build files, DI, and minimal code for 2 endpoints. |