

SpecVerse Ecosystem: Component Summaries

Detailed summaries of the four supporting repositories, written as building blocks for inclusion in a v2 introduction document.

1. specverse-app-demo — The Runtime Interpreter

What It Is

A full-stack application that **executes .specly files at runtime without compilation**. Load a YAML specification and instantly get a working application with REST API, WebSocket events, AI-assisted spec editing, web UI, and in-memory database — all dynamically generated from the spec.

This is not code generation. It's runtime interpretation: the specification *is* the application.

How It Works

```
blog.specly
  ↓
SpecVerseParser (@specverse/lang)
  ↓
DynamicSchemaRegistry (extracts models, controllers, events, services)
  ↓
RuntimeEngine
  ├── DynamicControllerEngine → REST API (CURVED operations)
  ├── BehaviorInterpreter → 15 convention patterns + Quint guard evaluation
  ├── LifecycleManager → State machines
  ├── EventBus → Pub/sub (1000 event history)
  ├── DynamicModelStore → In-memory database (auto-ID, auto-values)
  ├── AIService → Claude Code CLI integration (spawn per turn, streaming)
  └── WebSocket → Real-time events + AI streaming to browser
```

What Users See

Nine tabs in the browser:

Tab	Function
Views	Custom dashboards, lists, detail views, and forms defined in the spec. Pattern-based rendering with navigation between views.
Models	CRUD interface — create, edit, delete entities. Date pickers, relationship dropdowns, lifecycle state transitions. Auto-generated controllers for models without explicit ones.
Services	Real-time service operation monitoring.
Events	Live stream of every event in the system — entity created, state transitioned, spec reloaded.
Diagrams	Auto-generated Mermaid diagrams — ER, lifecycle, architecture, event flow, deployment topology.
3D Graph	Interactive Three.js visualisation of the architecture. Rotate, zoom, click nodes for details. Resizable sidebar with layer controls.
Specly	View and live-edit the raw .specly source with folding, validation, save (writes to disk + reloads runtime), and hot reload.
AI	Claude Code integration — type a prompt, Claude generates/modifies the .specly, click "Apply to Spec" to hot-reload. Shows system prompt and full prompt sent for transparency. Chat history persists across page reloads.
Help	Built-in SpecVerse documentation.

Multi-Server Mode

The Server Manager (port 9000) orchestrates multiple spec servers simultaneously:

- Browse filesystem for existing specs
- **Create New Spec** from engine-realize templates (with fallback)
- Each spec runs on its own port (3050, 3051, ...)
- Start/stop/restart individual servers

Behavior Execution

The runtime interprets L3 behaviors at runtime:

- **15 convention patterns** — CRUD, validation, lifecycle transitions, relationship management
- **117 Quint guards** — transpiled to TypeScript, evaluated at runtime as preconditions
- **Controller engine** — preconditions → steps → postconditions → events pipeline
- **Schema registry** — clears and rebuilds on spec reload for consistent state across all panes

Architecture

- **Backend:** Express + WebSocket (ws) + TypeScript
- **Frontend:** React + Vite + Tailwind + React Query
- **AI:** Claude Code CLI via spawn with `--print --output-format stream-json --resume`
- **3D Visualisation:** Three.js via layered-3d-graph library (separate repo, synced via build script)
- **Version:** 0.1.0

Why It Matters for the Introduction

The runtime interpreter proves a key claim: a .specly specification is **precise enough to execute directly**. You don't need to generate code first - the spec itself contains enough information to create a working application with CRUD operations, state machines, events, and UI. This makes the specification a first-class runtime artifact, not just a design document.

It also provides the **fastest possible feedback loop** for specification authors: write spec → see running app → modify spec → hot reload → see changes. No build step, no code generation, no deployment. The AI pane adds a second loop: describe what you want → Claude generates the spec → apply → see the running app.

Suggested Introduction Section

Live Specification Interpreter

SpecVerse specifications aren't just design documents - they're executable. The runtime interpreter loads a .specly file and creates a complete working application in seconds: REST API, WebSocket events, web interface, state machines, and relationship handling - all derived dynamically from your specification at runtime.

This provides the fastest possible feedback loop: write a spec, see it running, modify it, watch the application update in real time. No build step. No code generation. The specification is the application.

The integrated AI pane adds a second loop: describe what you want in natural language, Claude generates the .specly, click Apply, and the running application updates immediately. Both the system prompt and full prompt are visible for complete transparency.

The interpreter includes a 3D architecture visualisation, live event streaming, auto-generated diagrams, behavior execution with Quint guards, and a multi-server manager for running several specifications simultaneously.

2. specverse-demo-ai — The AI Testing Laboratory

What It Is

A comprehensive test suite proving that SpecVerse's AI workflow actually works. It validates that AI systems can reliably generate valid .specly specifications from natural language requirements (Pillar 2) and extract specifications from existing codebases (Pillar 3), with measurable quality metrics.

The Testing Progression

Three generations of testing, each more sophisticated:

Version	Approach	What It Proved
v1 (guesthouse-test)	Manual four-way comparison	Same domain produces 4x-7.6x expansion depending on scale
v2 (guesthouse-test-v2)	AI orchestration with custom bash	Create and analyse operations work with session management
v3 (guesthouse-test-v3)	Native SpecVerse sessions + automated framework	Full test automation with metrics, reports, and prompt version comparison

Four Test Operations

Each maps to a Four Pillars capability:

Operation	Pillar	Input	Output	Status
demo-create	Pillar 2	Simple requirements (40 lines)	Valid spec (~200 lines, 4x)	Working
pro-create	Pillar 2	Enterprise requirements (80 lines)	Valid spec (~3,600 lines, 7.6x)	Working
demo-analyse	Pillar 3	Clean demo codebase	Extracted spec	In progress
pro-analyse	Pillar 3	Production codebase	Extracted spec	In progress

The Scale Recognition Story

The most striking result: the same language specification format handles radically different scales:

Personal (demo-create):

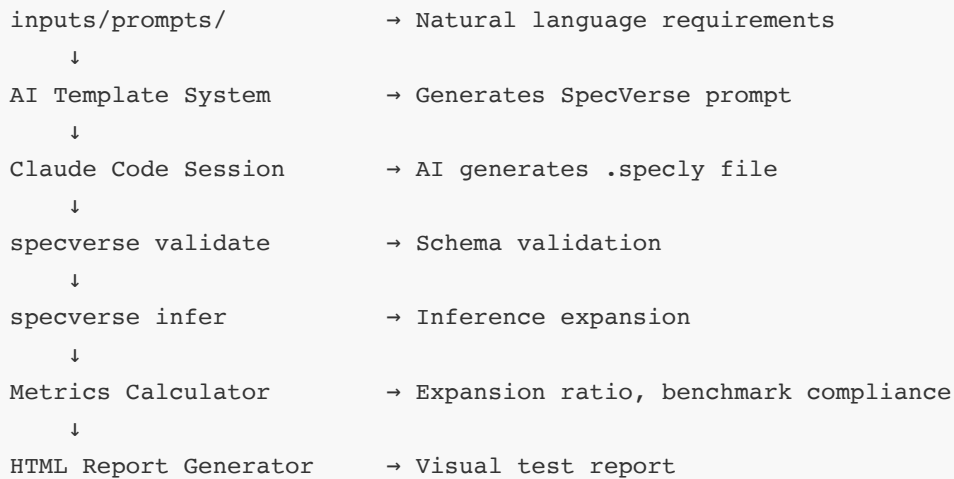
40 lines of requirements → 200 lines of spec (4x)
Single property, basic bookings, no auth

Enterprise (pro-create):

80 lines of requirements → 3,600+ lines of spec (7.6x)
Multi-tenancy, RBAC, 50+ countries, GDPR/PCI-DSS,
payment processing, occupancy analytics, integrations

The AI inference engine doesn't just produce more lines - it recognises the **kind** of system being described and generates architecturally appropriate patterns (multi-tenancy only when needed, RBAC only for enterprise, international support only when multiple countries are mentioned).

Test Framework Architecture



Session management uses native SpecVerse commands (`spv session create/submit/process`) with 98% token savings via schema caching - down from 245 lines of custom bash to 3 wrapper scripts.

Key Technical Details

- **Prompt versions:** v1 through v7+, with comparison infrastructure
- **Metrics:** Expansion ratio, validation pass rate, benchmark compliance ($\pm 20\%$ tolerance)
- **Codex fallback:** Alternative LLM provider for sandbox testing
- **Output per test:** config.yaml, spec.specly, validation.json, metrics.json, report.html

Current State

- **Working:** demo-create, pro-create with validate-fix loop
- **In progress:** demo-analyse, pro-analyse prompts need updating
- **Not started:** materialise, realize operations
- **Infrastructure:** Complete and automated

Why It Matters for the Introduction

This repository provides the **hard evidence** for SpecVerse's AI claims. Without it, "AI generates specs from requirements" is marketing. With it, there are measurable metrics: 4x expansion at personal scale, 7.6x at enterprise scale, 100% validation pass rate, repeatable across prompt versions. It also demonstrates the validate-fix loop pattern that makes AI generation reliable.

Suggested Introduction Section

Measured AI Performance

SpecVerse's AI capabilities aren't aspirational - they're measured. The test laboratory validates that AI systems can generate valid specifications from natural language requirements, with quantified results:

- **Personal scale**: 40 lines of requirements → 200 lines of validated spec (4x expansion)
- **Enterprise scale**: 80 lines of requirements → 3,600+ lines of validated spec (7.6x expansion) with multi-tenancy, RBAC, international support, and compliance patterns

The validate-fix loop ensures 100% schema compliance: generate → validate → fix errors → re-validate until the specification passes. Combined with session caching (98% token savings), this makes AI-assisted specification development both reliable and economical.

3. specverse-lang-doc — The Documentation Platform

What It Is

A production-grade Docusaurus v3 documentation site with 136+ pages covering the complete SpecVerse language, tooling, and ecosystem. Automatically synchronised from the specverse-lang repository.

Content Inventory

Section	Pages	Covers
Getting Started	7	Introduction, installation, quick start, architecture, AI integration, progressive examples, troubleshooting
Language Reference	18	Complete syntax: models, controllers, services, events, views, imports, types, lifecycles, profiles, AI inference engine
Examples	60+	24 core examples across 15 categories, each with .specly source + .mdx documentation
Registry	6	Platform overview, getting started, publishing, discovery, CLI integration, API reference
Reference	15	CLI reference, JSON schemas, glossary, build system, test framework, golden rules
Tools	8	Parser, validator, VSCode extension, language server, diagram generator, MCP server, app demo
Manifests	6	Structure, examples, implementation patterns
Guides	8	Validation, manifests, integration, patterns, CI/CD

Example Organisation

55 .specly source files paired with 60+ .mdx documentation pages, progressing from fundamentals to complex architectures:

01-fundamentals/	→ Models, lifecycles, behaviors, relations (5 examples)
02-profiles/	→ Dynamic composition (2 examples)
03-architecture/	→ Controllers, services, events, views, inference (8 examples)
04-domains/	→ E-commerce, organisation management (2 examples)
05-meta/	→ Self-describing specs, build systems (2 examples)
06-deploy/	→ Basic → Kubernetes → Docker → cloud (6 examples)
07-legacy/	→ v2.x migration patterns (3 examples)
08-comprehensive/	→ All features combined (1 example)
09-15/	→ Service layer, API, views, tools, diagrams, end-to-end (31 examples)

Each example includes metadata: difficulty level, prerequisites, estimated time, concepts introduced.

Sync Mechanism

A 515-line sync script (`scripts/sync-from-specverse-lang.js`) keeps documentation in sync with the source repository:

1. Copies .mdx documentation from specverse-lang's generated output
2. Syncs .specly source files from examples/

3. Copies auto-generated Mermaid diagrams
4. Merges sidebar configuration
5. Converts numbered directory references to clean paths
6. Cleans up outdated files
7. Falls back gracefully when specverse-lang isn't available (CI/CD safe)

Technical Details

- **Framework:** Docusaurus 3.0.1 with Mermaid theme
- **Custom components:** Mermaid renderer (SSR-safe), example loader, diagram components
- **Build:** Succeeds in ~8 seconds, 1 expected warning
- **Deployment:** Vercel-ready
- **Version:** @specverse/documentation 3.2.0

Audience Coverage

Audience	Entry Point	Path
Beginner	Introduction → Quick Start → Fundamentals examples	2-3 hours
Developer	Quick Start → Architecture → Services & Events examples	1-2 hours
Architect	Introduction → Component Architecture → Advanced examples	1-2 hours
DevOps	Deployment docs → Deployment examples → Registry publishing	1 hour

What's Strong

- Progressive learning path with 24 carefully structured examples
- Complete language reference (18 pages covering all constructs)
- Rich visualisation (15+ Mermaid diagram types integrated)
- Automatic sync from source repository
- Multiple audience support

What's Sparse

- Migration guides from v2.x (brief overview only)
- Advanced security patterns
- Performance optimisation guidance
- Video/interactive tutorials

Why It Matters for the Introduction

The documentation site is the primary entry point for anyone evaluating SpecVerse. Its quality directly affects adoption. The progressive examples (24 of them, from "here's a model" to "here's a full enterprise system") are the strongest onboarding asset. The automatic sync ensures documentation stays current with the language without manual maintenance.

Suggested Introduction Section

Documentation & Learning

136+ pages of documentation cover the complete SpecVerse ecosystem, from a 5-minute quick start to deep reference material on every language construct.

24 progressive examples guide learners from basic models through profiles, architecture patterns, domain modeling, deployment, and legacy migration. Each example pairs a `.specly` source file with generated documentation, so you can read the spec and understand what it produces.

The documentation automatically synchronises from the language repository, ensuring it stays current as the language evolves.

4. specverse-lang-registry — The Community Library Platform

What It Is

A production-deployed community registry for sharing and discovering reusable SpecVerse specifications. Think npm for `.specly` files. Monorepo containing API server, web frontend, and CLI - all designed from a SpecVerse specification (`spec/registry.specly`, 719 lines).

Architecture

```
specverse-lang-registry/
├── spec/registry.specly      ← Single source of truth (719 lines)
├── apps/
│   ├── api/                ← Fastify REST API (deployed on Vercel)
│   ├── web/                ← React browse/publish UI
│   └── cli/                ← @specverse/reg npm package
├── packages/
│   ├── shared/             ← TypeScript types
│   ├── ui/                 ← Reusable components
│   └── client/             ← API client
└── generated/              ← Auto-generated docs & diagrams from spec
```

What Users Can Do

Discover libraries:

```
specverse lib search authentication    # Search by keyword
specverse lib search --tags auth,oauth # Search by tags
specverse lib info @specverse/auth    # Get library details
specverse lib tags                    # Browse all categories
```

Publish libraries:

```
specverse-reg login                    # GitHub OAuth device flow
specverse-reg publish ./my-library.specly --tag auth --access public
```

Use libraries in specs:

```
import:
- from: "@specverse/auth"
  select: [User, AuthController]
- from: "@specverse/commerce"
  select: [Product, Order, OrderItem]
```

Platform Components

API Server (Fastify, 1,914 lines in app.ts):

- 25+ endpoints covering auth, libraries, search, tags, instance factories
- GitHub OAuth (device flow for CLI, standard for web)
- Server-side .specly validation via `specverse validate`
- PostgreSQL via Prisma, optional Redis caching
- Download tracking, star system, security scanning framework

Web UI (React + Vite + Tailwind):

- Browse libraries with type/category/tag filtering
- Library detail pages with version history and README
- Publish interface with drag-and-drop upload and dry-run validation
- Device auth flow for CLI authentication

CLI (@specverse/reg, published to npm):

- 8 commands: login, whoami, publish, unpublish, search, info, star, unstar
- Standalone - no SpecVerse dependency required
- Token stored in `~/.specverse/registry.json`

Database

PostgreSQL with Prisma. 9 models:

- **User** (GitHub OAuth), **Organisation**, **Library**, **LibraryVersion** (full .specly content stored)

- **Tag** (many-to-many with Library), **LibraryDependency**, **Download** tracking
- **SecurityScan**, **AnalyticsEvent**, **InstanceFactoryMeta** (v3.6.0 - capability-based discovery)

Current Content

- Live deployment at `https://specverse-lang-registry-api.vercel.app`
- Seed libraries: @specverse/auth, @specverse/commerce, @specverse/rest-api, @specverse/event-driven, domains-company, domains-retail
- Tag system active with categories: business, domains, authentication, api, primitives, deployment
- 49 comprehensive API tests passing
- Instance factory metadata system (v3.6.0) enabling capability-based factory discovery

Integration with specverse-lang

The CLI's `specverse lib` commands query the registry API directly:

- Import resolution checks registry before local files
- Content retrieved as raw .specly, fed to parser
- Results cached locally (5-10 minute TTL)
- Graceful fallback when registry unavailable

What's Specification-Designed vs Code-Generated

The registry was **designed from** `spec/registry.specly` (719 lines) but **implemented manually**. The spec serves as the architecture document:

- 4 models defined in spec → 9 database models implemented (extended)
- 4 controllers defined → 25+ endpoints implemented (richer than spec)
- 4 services defined → not yet implemented as separate service layer
- 7 events defined → not yet implemented

This is itself a data point about the maturity of the realize pipeline: the spec was used for design, but the jump to running code was manual.

Why It Matters for the Introduction

The registry demonstrates the **composability** value proposition. Instead of every project defining User, Auth, Organisation from scratch, you import proven, validated patterns. It also demonstrates that SpecVerse isn't just a solo tool - it has community infrastructure for sharing and reuse.

The registry itself being designed from a .specly spec is a nice meta-story: SpecVerse used to design the platform that hosts SpecVerse libraries.

Suggested Introduction Section

Community Library Registry

The SpecVerse Registry is a community platform for sharing reusable specifications – authentication patterns, e-commerce models, deployment templates, domain libraries. Think npm for .specly files.

```
```yaml
import:
 - from: "@specverse/auth"
 select: [User, AuthController]
```

Instead of defining common patterns from scratch, import proven, validated components. The registry includes tag-based discovery, version management, GitHub authentication, and full CLI integration via `specverse lib search`.

The registry platform itself was designed from a SpecVerse specification (719 lines), serving as both the community hub and a real-world validation of specification-driven architecture.

```

Cross-Cutting Observations for the Introduction v2

The Ecosystem Tells a Complete Story

These four repositories, combined with specverse-lang, cover the full
lifecycle:
```

Write spec (specverse-lang)

↓

Validate it works (specverse-app-demo — runtime interpreter)

↓

Prove AI can generate it (specverse-demo-ai — test laboratory)

↓

Share it with others (specverse-lang-registry — community platform)

↓

Learn how to use it (specverse-lang-doc — documentation)

### ### What's Production-Ready vs Emerging

Component	Maturity	Evidence
Language + Parser + Inference	Production	1,901 tests, sub-5ms
Documentation	Production	136+ pages, auto-sync
Runtime Interpreter	Beta	9-tab UI, AI integration, behavior execution
Registry	Production	Deployed, real content, 49 tests

AI Create (Pillar 2)	Validated	4x-7.6x measured expansion	
AI Analyse (Pillar 3)	Early	Prompts need updating	
AI Materialise (Pillar 4)	Not started	Framework ready	

### ### The Meta-Story

The registry was designed from a .specly spec. The runtime interpreter loads .specly files and creates apps from them. The test lab generates .specly files with AI and validates them. The documentation site syncs its examples from .specly source files. Everything in the ecosystem revolves around the .specly file as the central artifact. This consistency is the strongest argument for the communication protocol thesis.