RFC: Markdown → Interactive UI Compilation Framework

This document defines a framework for compiling Markdown documents into an Intermediate Representation (IR) that renders into interactive, visually rich user interfaces. The primary goal is to optimize communication between Large Language Models (LLMs) and humans by leveraging text for machines and visual cognition for humans.

1. High-Level Concept

The system introduces a three-layer architecture: Authoring (Markdown), Compilation (IR), and Rendering (UI). LLMs generate structured Markdown annotated with UI semantics. A compiler translates this into a deterministic IR. A JavaScript runtime renders the IR into interactive, animated, and explorable information UIs.

2. Core Abstractions

Block: atomic renderable units (text, graph, table). Component: parameterized interactive block with schema. Reference: typed links between blocks enabling cross-navigation. Patch: incremental IR update for whiteboard sessions.

3. Markdown Authoring Model

Extensions are expressed via fenced blocks (ui:graph, ui:table, ui:relation). These blocks contain YAML payloads validated against component schemas. This ensures LLM-friendly generation, diffability, and repository storage.

4. Intermediate Representation (IR)

The IR is a JSON document describing structure, components, layout hints, and interaction metadata. It is deterministic, versioned, and forward-compatible. IR supports patch-based updates for live sessions.

5. Rendering Runtime

A JavaScript/TypeScript runtime renders IR using React-compatible components. The runtime provides layout, theming, animation orchestration, and plugin registration.

6. Initial UI Tooling

graph: DAGs, flowcharts, mind maps, relational diagrams. table: sortable, filterable, aggregatable tables. chart: line, bar, area, financial OHLC. relation: entity-relationship diagrams via declarative YAML. timeline: event sequences and system evolution. callout, tabs, steps: narrative structuring tools.

7. Graph & Relation Abstraction

A unified relational abstraction defines nodes, edges, types, and layout semantics. Specific diagrams (ER, mind maps, architectures) derive from this base by constraining schemas.

8. Visualization Libraries

D3.js is recommended for graph layout, animation, and low-level control. Chart.js or ECharts may be used for standard charts. Layout engines include Dagre, ElkJS, and Force simulations.

9. Whiteboard Mode & MCP Integration

An MCP (Model Context Protocol) exposes tools for LLMs to create, modify, and patch IR documents. LLMs interact via high-level commands (add_node, link_entities). Sessions operate on IR patches and can be committed back into Markdown.

10. Use Cases

System architecture documentation. LLM-assisted design reviews. Interactive onboarding docs. Explaining ML pipelines, financial models, and abstract systems. Live collaborative reasoning sessions between humans and LLMs.

11. Storage & Versioning

Markdown remains the source of truth in Git. IR is derived and ephemeral, except during live sessions. Commits convert IR state back into canonical Markdown.

12. Future Work

Bidirectional Markdown↔IR normalization. Collaborative multi-user sessions. Export to PDF/Slides. Domain-specific component packs.