Bitcoinology v2 — Product Requirements Document

Version: 1.0
Date: 2026-02-12
Author: Max Power ⚡ — Belief Engines
Status: Master Document

> *"The Matrix, but orange."*
> Every search is visible work, not a loading spinner. Every belief is a collectible card. Every speaker is an RPG character. This isn't a dashboard — it's a game you want to play.

Table of Contents

Part 1: Vision & Strategy

Part 2: Design System

Part 3: User Experience Flows

Part 4: Component Specifications

- **Belief Cards**

- **Speaker System**

- **World Building**

Part 1: Vision & Strategy

1.1 Executive Summary

Bitcoinology is a gamified knowledge exploration platform that transforms Bitcoin podcast transcripts into a searchable, collectible, and shareable belief graph. Think Obsidian meets Pokémon — AI-powered search extracts atomic beliefs from 1000+ hours of podcast content, presents them as collectible cards, and lets users explore how speakers' worldviews connect, conflict, and evolve.

v1 proved the core engine works: vector search over belief embeddings, AI-powered multi-lens analysis (Playbook), speaker graphs, and streaming agentic workflows via Motia. But v1 looks like a developer tool — generic spinners, shadcn defaults, chat-style results.

v2 transforms the experience. Every search becomes a visible, animated pipeline. Every belief is a collectible card with pixel art and RPG mechanics. Every speaker is a character with tiers, stats, and evolving avatars. The app doesn't just find information — it makes finding information *fun*.

The vision: **"The Matrix, but orange."** Matrix rain background, Bitcoin orange accents, retro arcade aesthetics, and a game you don't want to stop playing.

1.2 Design Philosophy

Retro Arcade × Bitcoin

The entire UI language draws from NES/SNES-era gaming:

- **Mike Tyson's Punch-Out** — The search loading animation: pixel characters training on an orange track
- **Final Fantasy inventory screens** — The results grid: collectible belief cards as loot
- **Chrono Trigger character portraits** — Speaker profiles: RPG character sheets with stats and skill bars
- **Contra / Double Dragon** — Future loading animation variants
- **Pokémon rarity system** — Speaker tier system: Common → Rare → Epic → Legendary

Why Gamification?

1. **Engagement** — Watching a workflow tree build is genuinely fun, like watching a CI pipeline
2. **Trust** — Transparent agent steps show *exactly* how answers are assembled
3. **Retention** — Collectible cards and speaker tiers create progression loops
4. **Virality** — Beautiful social embed cards are growth vectors (every share = a new user)
5. **Education** — The RPG metaphor teaches Bitcoin concepts through exploration, not lectures

Design Principles

Principle	Meaning
Show the work	Every AI step is visible — no black boxes
Everything connects	Tap any node to go deeper — infinite rabbit holes

Principle	Meaning
Cards are currency	Beliefs are collectible, shareable, composable
Earn your place	Speakers level up through contribution, not clout
Game, not dashboard	Delight > efficiency. Fun > corporate

1.3 User Personas

🔬 The Bitcoin Researcher

"I want to find what Saylor actually said about nation-state adoption, with sources." - Needs: precise search, source citations, ontology navigation - Uses: Oracle search, expanded belief cards, speaker graphs - Success: finds the exact quote with episode + timestamp in under 10 seconds

🎮 The Casual Explorer

"I'm curious about Bitcoin but don't know where to start." - Needs: guided exploration, visual appeal, low barrier to entry - Uses: Playbook guided topics, card browsing, tier discovery - Success: spends 20+ minutes exploring without feeling lost

🏗️ The Community Contributor

"I have my own take on Bitcoin's monetary policy and I want to share it." - Needs: belief creation tools, research attachment, community validation - Uses: Community belief cards, voting, sharing - Success: creates a belief card that gets 50+ agrees

🎙️ The Podcast Speaker

"I want to see how my ideas connect to the broader Bitcoin discourse." - Needs: speaker profile, belief network, tier progression - Uses: RPG character sheet, force graph, connection scores - Success: sees their tier upgrade after a new episode drops

1.4 Entity Model — The Game Metaphor

Everything in Bitcoinology maps to a game concept:

Entity	Game Metaphor	Color	Description
🃏 **Beliefs**	Cards (collectible)	Orange `#F7931A`	Atomic claims extracted from podcasts. Collectible, shareable, inspectable. The core unit of the game.
👤 **People**	Characters (RPG)	Orange `#F7931A`	Speakers with stats, tiers, skill bars, and evolving pixel avatars. Character sheets.
🌐 **Domains**	Skill Trees	Orange `#F7931A`	Topic areas (Sound Money, Game Theory, Technology). Belief clusters within the ontology.

Entity	Game Metaphor	Color	Description
⚔️ **Events**	Arenas / Stages	Varies	Podcast episodes, conferences, debates. Where beliefs were spoken. Boxing rings, colosseums.
🏰 **Organizations**	Guilds / Factions	Varies	Companies, podcast networks, think tanks. Teams with collective worldviews and allegiances.
🃏 **Community Beliefs**	Crafted Cards	Teal `#00E5CC`	User-created beliefs backed by research and validated by community votes.

The loop: Cards come FROM arenas and are held BY guild members. Users MINE speaker beliefs through search and CRAFT community beliefs through creation. Everything connects.

1.5 Success Metrics

Metric	Target	How Measured
Search-to-result time	< 5s (Oracle), < 30s (Jackal)	SSE event timestamps
Cards viewed per session	> 10	Analytics
Session duration	> 5 minutes	Analytics
Social shares per week	> 50	OG image generation count
Community beliefs created per week	> 20	Supabase count
Speaker graph explorations per session	> 3 nodes deep	Click tracking
Core Web Vitals (LCP)	< 2.0s	Lighthouse
User retention (7-day)	> 30%	Analytics

Part 2: Design System

2.1 Color Tokens

Token	Hex	Usage
`--color-bg-primary`	`#0A0E1A`	App background, card interiors
`--color-bg-secondary`	`#111827`	Elevated surfaces, panels
`--color-bg-card`	`#1A1F2E`	Card body fill
`--color-accent-orange`	`#F7931A`	Bitcoin orange — borders, highlights, CTAs
`--color-accent-orange-glow`	`#F7931A80`	Orange glow (50% opacity for `box-shadow`)
`--color-accent-orange-dim`	`#C47615`	Pressed/active orange

Token	Hex	Usage
`--color-accent-teal`	`#00E5CC`	Community elements — borders, badges
`--color-accent-teal-glow`	`#00E5CC60`	Teal glow
`--color-accent-gold`	`#FFD700`	Legendary tier border + text
`--color-accent-purple`	`#9B59B6`	Epic tier glow + energy
`--color-accent-blue`	`#3498DB`	Rare tier glow
`--color-accent-gray`	`#6B7280`	Common tier border
`--color-text-primary`	`#FFFFFF`	Headings, primary text
`--color-text-secondary`	`#9CA3AF`	Subtitles, metadata
`--color-text-orange`	`#F7931A`	Speaker names, stats numbers
`--color-success`	`#22C55E`	Checkmarks, completed steps
`--color-error`	`#EF4444`	Failed steps, errors
`--color-polarity-for`	`#22C55E`	"For" polarity indicator
`--color-polarity-against`	`#EF4444`	"Against" polarity indicator
`--color-polarity-neutral`	`#9CA3AF`	"Neutral" polarity indicator

2.2 Typography

Token	Font Family	Weight	Size	Usage
`--font-pixel-heading`	`"Press Start 2P"`	400	16-24px	Screen titles, "BELIEFS FOUND", "LEGENDARY"
`--font-pixel-body`	`VT323`	400	18-24px	Card body text, quotes, stats
`--font-pixel-small`	`VT323`	400	14-16px	Metadata, labels, timestamps
`--font-ui`	`Inter`	400/600/700	14-18px	Search bar input, system UI, non-pixel contexts

Loading: Google Fonts — `Press Start 2P` and `VT323` via `next/font/google` with `display: swap`. Preload both in `<head>`.

2.3 Spacing Grid

Base unit: **4px**. All spacing is multiples of 4.

Token	Value	Usage
`--space-1`	4px	Tight inline gaps
`--space-2`	8px	Icon-to-text, compact padding
`--space-3`	12px	Card internal padding
`--space-4`	16px	Standard padding, card gaps
`--space-6`	24px	Section gaps
`--space-8`	32px	Major section spacing

2.4 Responsive Breakpoints

Name	Width	Columns	Card Width
Mobile	320–767px	1	100% (max 390px)
Tablet	768–1023px	2	~340px
Desktop	1024px+	3–4	~300px

Layout approach: Single-pane app. No route changes — all navigation is panel swaps with transitions. Mobile-first CSS with `min-width` media queries.

2.5 Border & Glow Effects

Effect	CSS	Usage
Orange card border	`border: 2px solid #F7931A`	Speaker belief cards
Orange glow	`box-shadow: 0 0 12px #F7931A80, inset 0 0 6px #F7931A40`	Active/hover belief cards
Teal card border	`border: 2px solid #00E5CC`	Community belief cards
Teal glow	`box-shadow: 0 0 12px #00E5CC60`	Community card hover
Gold legendary border	`border: 2px solid #FFD700; box-shadow: 0 0 20px #FFD70060`	Legendary tier cards
Purple epic glow	`box-shadow: 0 0 16px #9B59B680`	Epic tier cards
Blue rare glow	`box-shadow: 0 0 12px #3498DB60`	Rare tier cards
Pixel border (8-bit)	Use `<PixelBorder>` component with image-based corners	Pixel art variant cards
Lightning pulse	`animation: pulse-glow 1.5s ease-in-out infinite`	Active workflow nodes

2.6 Animation Timing Standards

Animation	Duration	Easing	Usage
Lightning flash	500ms	`ease-out`	Search complete → bolt strike
Card flip/expand	300ms	`cubic-bezier(0.4, 0, 0.2, 1)`	Card face → expanded
Workflow node complete	200ms	`ease-in`	Checkmark appear
Glow pulse	1500ms	`ease-in-out` (infinite)	Active pipeline glow
Sprite frame	150ms per frame	`steps(N)`	Pixel character animations
Fade in (results)	400ms staggered (50ms per card)	`ease-out`	Results grid population
Panel slide	250ms	`cubic-bezier(0.4, 0, 0.2, 1)`	Pane transitions

2.7 Shared Components

2.7.1 `<PixelBorder />`

Props: `variant: 'orange' | 'teal' | 'gold' | 'purple' | 'blue' | 'gray'`, `glow?: boolean`, `animated?: boolean`, `children`

Renders a container with 8-bit style corner ornaments and side borders using CSS `border-image` or 9-slice background image. When `glow` is true, adds the matching `box-shadow`. When `animated`, the border has a subtle shimmer animation (2s cycle).

2.7.2 `<PixelHealthBar />`

Props: `value: number` (0–100), `label?: string`, `color?: 'orange' | 'teal' | 'green'`, `size?: 'sm' | 'md' | 'lg'`

Segmented retro health bar. Bar is divided into discrete pixel blocks. Fill color defaults to `--color-accent-orange`. Percentage text in `VT323` right-aligned. Sizes: sm=16px h, md=24px h, lg=32px h.

2.7.3 `<PixelAvatar />`

Props: `src: string`, `size: 'xs' | 'sm' | 'md' | 'lg' | 'xl'`, `tier?: TierLevel`, `borderColor?: string`

Renders speaker avatar in a pixel-art circular frame. Tier determines frame ornament complexity (Common=plain, Legendary=ornate with laser effects). Sizes: xs=32px, sm=48px, md=64px, lg=96px, xl=128px. Uses `image-rendering: pixelated` for pixel art sources.

2.7.4 `<PixelBadge />`

Props: `label: string`, `variant: 'orange' | 'teal' | 'gold' | 'purple'`, `icon?: ReactNode`

Small pill-shaped badge with pixel font text. Used for tier labels, ontology badges, topic tags.

2.7.5 `<PixelIcon />`

Props: `name: string` (from icon set), `size: number`, `color?: string`

16×16 or 32×32 pixel art icon sprites. Icon names: `ontology`, `connections`, `source`, `lightning`, `sword`, `scroll`, `microphone`, `share`, `search`, `brain`, `rank`, `synthesize`.

2.7.6 `<PixelButton />`

Props: `variant: 'primary' | 'secondary' | 'ghost'`, `size: 'sm' | 'md'`, `children`, `onClick`

Retro-styled button with pixel borders. Primary = orange fill, secondary = outlined, ghost = text-only. Press state shifts 2px down (like NES button press).

2.7.7 `<RetroSearchBar />`

Props: `value: string`, `onChange`, `onSubmit`, `placeholder?: string`, `isSearching?: boolean`

Orange-bordered input field with pixel-style search icon. When `isSearching`, text animates as "Searching beliefs..." with blinking cursor. Full width on mobile, max 600px on desktop. 48px min height for touch target.

Part 3: User Experience Flows

3.1 Search Flow — Loading → Strike → Tree → Results

The search experience is the hero flow. It transforms a mundane search into a spectacle.

Step 1: The Training Run (Loading)

Search Loading

User submits a query → full-screen retro 8-bit animated scene plays. Punch-Out training montage: two pixel characters running on an orange track with a Bitcoin logo in the background. "Searching beliefs..." in the search bar with blinking ellipsis. Loops while agents process.

Step 2: The Strike (Transition)

Lightning Strike

First SSE `step_start` event arrives → screen flashes white → massive orange lightning bolt cracks down the center. "BELIEFS FOUND" in arcade font. The bolt IS the data pipeline — not decoration. 500ms choreographed animation.

Step 3: The Workflow Tree (Pipeline)

Lightning Tree

The bolt becomes a glowing pipeline. Each node = a real agent step: SEARCH 🔍 → ANALYZE 🧠 → RANK ⚖️ → SYNTHESIZE 📝. Green checkmarks light up as each completes. Energy particles flow downward. For Jackal deep research, the bolt branches into parallel threads. Users watch their answer being built in real-time.

Step 4: The Loot (Results)

Results Screen

Pipeline completes → belief cards land in a grid. Orange-bordered tiles with quotes, mini graphs, pixel art borders. Feels like an RPG inventory screen — these are your collected beliefs, your loot from the search.

3.2 Card Interaction Flow — Face → Expanded → Share

Face — The Collectible

Belief Card Face

Tap any belief card in the results grid → the card is a self-contained unit with speaker avatar, quote, confidence health bar, and action icons.

Expanded — The Portal

Belief Card Expanded

Tap to inspect → left side shows ontology tree (Core Axiom → Worldview → Claim). Right side shows connected speakers who share or challenge this belief. Every node is tappable — rabbit hole by design.

Social Embed — The Growth Loop

Social Embed

Share on Twitter/X → beautiful OG card with @Bitcoinology branding, quote, speaker + episode, ontology badge. Every shared card is a growth vector — tap to land in the app on that exact belief.

3.3 Speaker Exploration Flow — Profile → Graph → Connections

Character Select

Speaker Profile

Tap a speaker name anywhere → RPG character select screen. Avatar with orange ring, stats (Episodes, Beliefs, Connections), belief breakdown bars, mini belief card thumbnails.

Deep Inspection

Speaker Expanded

Tap profile → full belief network as force-directed graph. Speaker node centered, belief clusters radiating. Sidebar shows top connections with agreement scores. Every node tappable.

The Character Sheet

Pixel RPG Sheet

The crown jewel. Legendary portrait with laser eyes, ornate pixel frame. Full stat block. Skill bars. This IS a character select screen.

3.4 Community Participation Flow — Create → Research → Vote

Community Card

1. User creates their own belief card (teal border distinguishes from speaker-sourced)
2. Attaches research evidence (articles, clips, data) as pixel item icons
3. Community votes: ⚡ agrees / ⚔️ challenges
4. High-quality beliefs can merge into the main graph

Community cards are **CRAFTED** by users; speaker cards are **MINED** from podcast data.

3.5 Full User Flow Diagram

```
 Open App
  → Matrix rain + retro title screen
  → Search bar prominently centered

Type Search
  → Retro training animation plays (Punch-Out jog)
  → Lightning STRIKES the screen
  → Bolt becomes the pipeline tree
  → Steps complete top to bottom (SEARCH → ANALYZE → RANK → SYNTHESIZE)
  → Flash
  → Belief cards land as results (your loot)

Tap a Belief Card
  → Card expands: ontology tree + connected speakers
  → Tap a speaker → their RPG character sheet
  → Tap a connection → another belief card
  → Infinite exploration

Share a Card
  → Beautiful social embed generates
  → Someone taps it → lands in the app on that belief
  → Growth loop

Community Participation
  → User creates their own belief card (teal border)
  → Attaches research (articles, clips, data)
```

```
   → Community votes: agrees ⚡ / challenges ⚔
   → High-quality beliefs can merge into main graph
```

Part 4: Component Specifications

All 16 components with full visual specs, data requirements, interaction specs, acceptance criteria, and gap analysis.

Search Experience

4.1 Search Loading Animation

Screenshot Reference

Search Loading

Component Name

`<SearchLoadingAnimation />`

Description

Full-screen retro 8-bit animated scene that plays while agents process a search query. Shows two pixel characters running on an orange track (Punch-Out training montage style) with a Bitcoin logo in the background. The search bar shows "Searching beliefs..." at the top. Loops until search completes.

Visual Requirements

Property	Value
Background	`#0A0E1A` (dark navy)
Track color	`#F7931A` (Bitcoin orange) with white lane lines
Characters	Two 32×48px pixel sprites — runner (dark) + coach (orange jacket)
Animation type	Sprite sheet, 4-frame run cycle, 150ms/frame
Search bar text	`VT323` 20px, `#F7931A`, blinking ellipsis
Bitcoin logo	48×48px pixel art, upper-left, static
Mobile	Full viewport, characters centered vertically in lower 60%
Tablet/Desktop	Centered with max-width 500px, vertically centered

Sprite sheet spec: Each character has a 4-frame horizontal sprite sheet (128×48px total). CSS `steps(4)` animation. Coach runs slightly behind the runner.

Variant scenes (future): Randomly select from 3 scenes: Punch-Out jog (default), Contra run-and-gun, Double Dragon street walk. MVP ships with jog only.

Data Requirements

Data	Source	Notes
`isSearching: boolean`	Local state (Zustand)	Set true on search submit
SSE stream connection	Motia `POST /chat` → `streamUrl`	Needed to know when to transition
`step_start` SSE event	Motia stream	First event triggers transition to lightning

Interaction Spec

- **Entry:** Plays immediately on search submit. Panel slides up from bottom (250ms).
- **During:** No user interaction — animation loops. Back button / escape cancels search.
- **Exit:** On first `step_start` SSE event, fade to white (100ms) then trigger `<LightningStrikeTransition />`.
- **Error:** If no SSE events arrive within 10s, show "Still searching..." text. At 30s, show retry button.

Acceptance Criteria

1. **Given** a search is submitted, **When** the animation starts, **Then** pixel characters animate at exactly 150ms per sprite frame with no stutter.
2. **Given** the animation is playing, **When** the first SSE `step_start` event arrives, **Then** the animation fades to white flash within 100ms.
3. **Given** the animation is playing on a 390px viewport, **Then** characters are centered and no content overflows horizontally.
4. **Given** no SSE events arrive within 10 seconds, **When** the timeout fires, **Then** "Still searching..." text appears below the search bar in `VT323` 16px.
5. **Given** the user taps back/escape during animation, **When** the action fires, **Then** the search is cancelled, SSE stream closed, and app returns to previous state.

Dependencies

- `<RetroSearchBar />` component
- Sprite sheet assets (runner + coach PNG)
- SSE stream connection from Motia `/chat` endpoint
- Zustand `isSearching` state

Gap Analysis

Feature	v1 Status	v2 Action
Search loading	Generic spinner (`src/components/playbook/skeleton.tsx`)	**Build from scratch** — pixel animation replaces skeleton

Feature	v1 Status	v2 Action
SSE connection	Exists (`@motiadev/stream-client-react`)	**Reuse** — hook into existing stream
Search state	Zustand store (`src/stores/app-store.ts`)	**Extend** with `isSearching` flag
Sprite assets	None	**Create** — pixel art sprite sheets

4.2 Lightning Strike Transition

Screenshot Reference

Lightning Strike

Component Name

`<LightningStrikeTransition />`

Description

Full-screen lightning bolt that cracks down the center of the screen when search results are ready. "BELIEFS FOUND" text appears in arcade font at the top. The bolt then splits at the bottom into branches that morph into the workflow tree nodes. This is a 500ms choreographed animation, not a loop.

Visual Requirements

Property	Value
Background	Black (`#000000`) during flash, fades to `#0A0E1A`
Lightning bolt	`#F7931A` with white (`#FFFFFF`) core, 60px wide at top tapering to 20px
Bolt glow	`box-shadow: 0 0 40px #F7931A` along path
"BELIEFS FOUND" text	`Press Start 2P` 24px mobile / 36px desktop, `#F7931A`
Crack lines at impact	Thin (`1px`) orange veins radiating from impact point, 8-12 branches
Bottom split	Bolt forks into 3-4 branches pointing to where workflow nodes will appear

Animation timeline (500ms total):

Time	Event
0ms	Screen flashes white (opacity 0→1→0 in 50ms)
50ms	Bolt begins drawing top-to-bottom (SVG path animation)
200ms	Bolt fully drawn, "BELIEFS FOUND" text fades in
300ms	Impact point: crack lines radiate

Time	Event
400ms	Bolt bottom forks into branches
500ms	Hold — transition to `<LightningWorkflowTree />` begins

Implementation: SVG `<path>` with `stroke-dasharray` + `stroke-dashoffset` animation. The bolt path should have 2-3 zig-zag segments (not straight). Use `requestAnimationFrame` for smooth 60fps rendering.

Data Requirements

Data	Source
`resultsReady: boolean`	SSE `step_start` event from Motia stream
`resultCount: number`	Optional — can show "23 BELIEFS FOUND"

Interaction Spec

- **Entry:** Triggered by `<SearchLoadingAnimation />` exit.
- **During:** No user interaction. Non-interruptible 500ms animation.
- **Exit:** At 500ms, cross-fade (200ms) into `<LightningWorkflowTree />`. The bolt's fork branches align with the workflow tree's first nodes.

Acceptance Criteria

1. **Given** search loading completes, **When** the transition plays, **Then** the bolt draws top-to-bottom in exactly 150ms (from 50ms to 200ms mark).
2. **Given** the animation plays, **When** "BELIEFS FOUND" appears, **Then** the text is horizontally centered and uses `Press Start 2P` at 24px on ≤767px viewports.
3. **Given** the full 500ms animation completes, **When** the workflow tree begins rendering, **Then** the bolt's fork points align within 8px of the first workflow node positions.
4. **Given** any viewport width 320–1440px, **Then** the bolt is vertically centered on screen with no horizontal overflow.

Dependencies

- `<SearchLoadingAnimation />` (provides transition trigger)
- `<LightningWorkflowTree />` (receives handoff)
- SVG bolt path asset (can be procedurally generated)

Gap Analysis

Does not exist in v1. Entirely new component. v1 has no visual transition between search submit and results.

4.3 Lightning Workflow Tree

Screenshot Reference

Lightning Tree

Component Name

`<LightningWorkflowTree />`

Description

A vertical pipeline visualization showing each agent processing step in real time. A glowing orange energy line runs top-to-bottom connecting circular nodes. Each node represents a pipeline step (SEARCH → ANALYZE → RANK → SYNTHESIZE). Nodes light up with green checkmarks as each completes. The bolt pulses with energy flowing downward. For Jackal deep research, the bolt branches into parallel threads.

Visual Requirements

Property	Value
Background	`#0A0E1A` with subtle radial gradient (darker at edges)
Energy line	4px wide, `#F7931A` , `glow: 0 0 12px #F7931A80`
Energy flow	Particle animation: small dots (4px) travel down the line at 200px/s
Node circle	64px diameter, `#1A1F2E` fill, 3px `#F7931A` border
Node icon	32×32px, orange (`#F7931A`) — magnifier, brain, scales, document
Node label	`Press Start 2P` 12px, `#FFFFFF` , right or left of node (alternating)
Checkmark	20px, `#22C55E` , appears top-left of node with 200ms pop-in
Active node	Border pulses (`animation: pulse-glow 1.5s infinite`), icon animates
Queued node	50% opacity, no glow
Failed node	Border `#EF4444` , × icon instead of checkmark

Node states:

State	Visual
`queued`	50% opacity, gray border, no glow
`active`	Full opacity, orange border pulsing, energy particles entering node
`complete`	Full opacity, green checkmark, static orange border
`failed`	Full opacity, red border, red × icon, retry button appears

Layout: - Mobile: Nodes stacked vertically, centered, 80px vertical gap between nodes - Tablet+: Same vertical stack, max-width 400px centered - Labels alternate left/right of the energy line for visual interest

Jackal parallel branching: When `task.enrich` runs parallel sub-queries, the single energy line splits into N branches below the SEARCH node, each with its own sub-node, then reconverges at RANK.

Data Requirements

Data	Source	Type
Pipeline steps	SSE events from Motia stream	Real-time
`step_start`	`{"type":"step_start","id":"vector_search","label":"🔍 Vector Search"}`	SSE
`step_progress`	`{"type":"step_progress","id":"vector_search","data":{"found":23}}`	SSE
`step_complete`	`{"type":"step_complete","id":"vector_search","duration_ms":340}`	SSE
`stream_token`	`{"type":"stream_token","id":"synthesis","token":"Michael"}`	SSE
Agent type	`task.search` / `task.playbook` / `task.enrich` / `task.advise`	SSE routing event

Step mapping by agent type:

Agent	Steps shown
Direct Search (Oracle)	SEARCH → RANK → SYNTHESIZE (3 nodes)
Playbook	SEARCH → ANALYZE (4 parallel lenses) → SYNTHESIZE (branching tree)
Jackal	SEARCH → DECOMPOSE → PARALLEL SEARCH (N branches) → CROSS-REF → SYNTHESIZE
Oracle (multi-perspective)	QUICK TAKE → DEEP ANALYSIS → STRATEGIC VIEW → SYNTHESIZE

Interaction Spec

- **Entry:** Cross-fades in from `<LightningStrikeTransition />` at the 500ms mark. First node is already `active`.
- **During:** Users watch. Each `step_complete` SSE event triggers: checkmark animation (200ms) → energy flows to next node → next node becomes `active`.
- **Tap on completed node:** Expands inline to show metadata (e.g., "23 beliefs found in 340ms"). 300ms expand animation.
- **Tap on failed node:** Shows retry button. Tap retry sends `retry` event to Motia.
- **Exit:** On final `step_complete`, hold for 500ms, then slide-transition to `<ResultsGrid />`.
- **Progress data:** If `step_progress` includes counts (e.g., `found: 23`), display as small badge on the active node.

Acceptance Criteria

1. **Given** the workflow tree is visible, **When** a `step_start` SSE event arrives for "vector_search", **Then** the SEARCH node transitions from `queued` to `active` within one animation frame.
2. **Given** the SEARCH node is `active`, **When** `step_complete` fires, **Then** a green checkmark pops in (200ms), energy particles flow to the next node, and the next node becomes `active` — all within 400ms total.
3. **Given** a Jackal deep research query, **When** the DECOMPOSE step completes with 3 sub-queries, **Then** the energy line visually splits into 3 parallel branches below the DECOMPOSE node.
4. **Given** a step fails, **When** `step_error` SSE event arrives, **Then** the node turns red, shows × icon, and a "Retry" `<PixelButton>` appears within 200ms.
5. **Given** all steps complete, **When** 500ms hold timer expires, **Then** the tree slides up and `<ResultsGrid />` slides in from below.

6. **Given** the tree is on a 390px viewport, **Then** all nodes are visible without horizontal scrolling and labels don't overflow.

Dependencies

- `<LightningStrikeTransition />` (entry)
- SSE stream from Motia (`@motiadev/stream-client-react`)
- `<ResultsGrid />` (exit target)
- `<PixelIcon />` for node icons

Gap Analysis

Feature	v1 Status	v2 Action
Workflow visualization	None — results appear after loading skeleton	**Build from scratch**
SSE streaming	`@motiadev/stream-client-react` exists, SSE events defined in architecture	**Reuse** stream client, may need new event types
Pipeline step tracking	Motia steps emit events (`src/app/api/query/route.ts`)	**Extend** — ensure all steps emit `step_start` / `step_complete` events

---

4.4 Results Grid

Screenshot Reference

Results Grid

Component Name

`<ResultsGrid />`

Description

Grid of belief cards displayed as search results. Orange-bordered tiles with quotes, a small graph visualization showing belief connections, and pixel-art decorative elements framing the grid. Search bar persists at top. Feels like an RPG inventory/loot screen.

Visual Requirements

Property	Value
Background	`#0A0E1A`
Grid layout	CSS Grid, `gap: 16px`
Mobile	1 column, cards full-width
Tablet	2 columns

Property	Value
Desktop	2–3 columns, max-width 1200px centered
Card tiles	Orange border (`2px solid #F7931A`), rounded corners `8px` , `#1A1F2E` fill
Quote text	`VT323` 18px, `#FFFFFF`
Search bar	Sticky top, `<RetroSearchBar />`
Decorative frame	Pixel art sidebar elements (visible on desktop only), `retro-ux/04` style
Mini graph	120×80px inline chart (belief connections) using SVG, `#F7931A` lines and dots
Empty state	"No beliefs found" in `Press Start 2P` 16px, centered, with sad pixel character

Card entrance animation: Cards fade in staggered — first card at 0ms, each subsequent +50ms delay, 400ms fade duration. Max 12 cards visible initially; "Load More" pixel button at bottom.

Data Requirements

Data	Source
`BeliefResult[]`	SSE `response.ready` event → `citations` + search results
Per-belief: `id` , `speaker_name` , `topic` , `atomic_belief` , `quote_text` , `polarity` , `polarity_confidence` , `podcast_slug` , `episode_slug` , `timestamp_start`	Motia response / Qdrant + DuckDB
Connection graph data (which beliefs relate)	`core_aggregations.parquet` via DuckDB-WASM
Synthesized answer text	SSE `stream_token` events → concatenated

API: `POST /chat` → SSE stream → `response.ready` event provides `answer` + `citations[]` . Each citation's `belief_id` used to fetch full belief data.

Interaction Spec

- **Entry:** Slides up from bottom as workflow tree slides up (250ms).
- **Tap card:** Opens `<BeliefCardExpanded />` — card zooms to fill panel (300ms).
- **Tap mini graph:** Navigates to ontology graph view centered on that belief cluster.
- **Scroll:** Infinite scroll, loads 12 cards per batch.
- **Pull-to-refresh (mobile):** Re-runs the search.
- **Search bar submit:** Clears grid, triggers new search flow (back to `<SearchLoadingAnimation />`).

Acceptance Criteria

1. **Given** search results arrive, **When** the grid renders, **Then** cards appear with 50ms stagger and each card's fade-in lasts 400ms.
2. **Given** 24 results exist, **When** the grid initially renders, **Then** only 12 cards are visible and a "Load More" button is present at the bottom.

3. **Given** a result card is tapped, **When** the card expands, **Then** the `<BeliefCardExpanded />` is shown within 300ms with no layout shift in the background grid.
4. **Given** results are on a 390px viewport, **Then** cards are single-column, full-width with 16px horizontal padding.
5. **Given** no results found, **Then** an empty state with "No beliefs found" text and pixel character is displayed centered.

Dependencies

- `<LightningWorkflowTree />` (entry transition)
- `<BeliefCardFace />` (card tiles)
- `<BeliefCardExpanded />` (tap target)
- `<RetroSearchBar />` (persistent top)
- SSE stream `response.ready` data
- DuckDB-WASM for connection graph

Gap Analysis

Feature	v1 Status	v2 Action
Results display	`src/components/search/belief-card.tsx` — basic card list	**Redesign** with pixel art grid, stagger animation
Search results	`src/components/search/chat.tsx` — streaming chat format	**Replace** chat-style with card grid
Graph mini-viz	`src/components/graph/speaker-graph.tsx` (full page)	**New** — inline mini-graph per card
Infinite scroll	Not implemented	**Build**

Belief Cards

4.5 Belief Card — Face

Screenshot Reference

Belief Card Face

Component Name

`<BeliefCardFace />`

Description

The collectible face of a belief card. Shows speaker avatar, the direct quote, speaker name + episode reference, a confidence meter as a retro health bar, and three action icons at the bottom (Ontology, Connections, Source). This is the primary card unit throughout the app.

Visual Requirements

Property	Value
Card size	300×400px default, responsive (min 280px, max 340px width)
Background	`#1A1F2E`
Border	2px solid `#F7931A`, rounded 8px
Outer glow	`box-shadow: 0 0 12px #F7931A80` on hover/active
Inner glow	`box-shadow: inset 0 0 20px #F7931A20` (subtle, always on)
Speaker avatar	`<PixelAvatar size="md" />` — 64px, top-left corner, 12px margin
Quote text	`VT323` 20px, `#FFFFFF`, max 4 lines, ellipsis overflow
Speaker name	`Press Start 2P` 12px, `#F7931A`
Episode ref	`VT323` 14px, `#9CA3AF` (e.g., "WBD #412")
Confidence bar	`<PixelHealthBar value={94} label="CONFIDENCE" />` — full width, 24px height
Action icons	3 icons in a row, 32px each, `#9CA3AF` default, `#F7931A` on tap
Icon labels	`VT323` 10px, `#9CA3AF` — "Ontology", "Connections", "Source"

Hover state (desktop): Card lifts 4px (`transform: translateY(-4px)`), glow intensifies. Transition 200ms.

Polarity indicator: Thin 3px colored bar at top of card: green for "for", red for "against", gray for "neutral".

Data Requirements

Field	Source	Column
Speaker avatar URL	Supabase `speakers` table or generated pixel avatar	`speakers.avatar_url`
Quote text	Qdrant / beliefs_summary.parquet	`quote_text`
Atomic belief	beliefs_summary.parquet	`atomic_belief`
Speaker name	beliefs_summary.parquet	`speaker_name`
Episode slug	beliefs_summary.parquet	`episode_slug`
Podcast slug	beliefs_summary.parquet	`podcast_slug`
Confidence	beliefs_summary.parquet	`polarity_confidence` (×100 for %)
Polarity	beliefs_summary.parquet	`polarity` ("for"/"against"/"neutral")
Ontology data	beliefs_summary.parquet	`core_axiom`, `worldview`, `topic`
Belief ID	beliefs_summary.parquet	`id`

Interaction Spec

Action	Behavior
Tap card body	Expand to `<BeliefCardExpanded />` (300ms zoom transition)
Tap Ontology icon	Expand card to show ontology tree inline
Tap Connections icon	Expand card to show connected speakers
Tap Source icon	Open source episode (external link or in-app if available)
Long press (mobile)	Show share menu (Copy link, Share to X, Save)
Hover (desktop)	Card lifts 4px, glow intensifies

Acceptance Criteria

1. **Given** a belief with `polarity_confidence: 0.94`, **When** the card renders, **Then** the health bar shows exactly 94% fill with "94%" label.
2. **Given** a quote longer than 4 lines at 20px VT323, **When** the card renders, **Then** the text is truncated with an ellipsis on the 4th line.
3. **Given** a belief with `polarity: "against"`, **When** the card renders, **Then** a 3px red bar appears at the top of the card.
4. **Given** the card is tapped, **When** the expand animation plays, **Then** the card transitions to `<BeliefCardExpanded />` within 300ms.
5. **Given** a desktop viewport, **When** hovering over the card, **Then** the card rises 4px and glow intensifies within 200ms.
6. **Given** a mobile viewport (390px), **Then** the card width is at least 280px and no content overflows.

Dependencies

- `<PixelAvatar />`
- `<PixelHealthBar />`
- `<PixelIcon />` (ontology, connections, source)
- `<PixelBorder />` (orange variant)
- Belief data from search results

Gap Analysis

Feature	v1 Status	v2 Action
Belief card	`src/components/search/belief-card.tsx` — shadcn Card, no pixel art	**Redesign** completely with pixel aesthetic
Confidence bar	Not shown in v1	**New** component
Avatar	Not shown on cards in v1	**Add** `<PixelAvatar />`
Polarity indicator	Not visualized in v1	**Add** colored bar
Ontology/Connections/Source icons	Not in v1 cards	**Add** action row

---

4.6 Belief Card — Expanded

Screenshot Reference

Belief Card Expanded

Component Name

`<BeliefCardExpanded />`

Description

Full expanded view of a belief card. Left side shows the original card content plus an ontology tree (Core Axiom → Worldview → Claim). Right side shows connected speakers who share or challenge this belief, with mini-cards linked by connection lines. Every node is tappable for further exploration.

Visual Requirements

Property	Value
Layout	Mobile: stacked (card top, ontology middle, connections bottom). Tablet+: two-column (card+ontology left, connections right)
Background	`#0A0E1A` full-panel overlay
Card section	Same as `<BeliefCardFace />` but larger (fills left column)
Ontology tree	Vertical tree: Core Axiom (top) → Worldview → Claim (bottom). Orange nodes, orange connecting lines
Ontology node	Rounded rect, `#1A1F2E` fill, `2px #F7931A` border, `VT323` 16px text
Ontology node label	`Press Start 2P` 10px, `#F7931A`, above node (e.g., "Core Axiom")
Connection lines	`1px dashed #F7931A60` connecting quote card to ontology, ontology to speakers
Connected speakers	Mini cards: 200×80px, avatar (32px) + name + agreement score
Speaker connection line	`1px solid #F7931A`, orange dot at junction, connecting speaker mini-card to the relevant ontology node
Close button	Top-right, `×` in pixel style, 44×44px touch target
Decorative corners	Orange pixel-art corner ornaments (from `belief-cards/02` mockup)

Ontology tree values from data:

```
Core Axiom:  belief.core_axiom
    ↓
Worldview:   belief.worldview
    ↓
Claim:       belief.atomic_belief
```

Data Requirements

Data	Source
All `<BeliefCardFace />` data	Already loaded
`core_axiom`	beliefs_summary.parquet
`worldview`	beliefs_summary.parquet
Connected speakers	DuckDB-WASM query: speakers who share the same `worldview` or `core_axiom`, ranked by `speaker_similarity` matrix from `core_aggregations.parquet`
Agreement score	Computed: cosine similarity between this belief's `weights[10]` and connected speaker's average `weights[10]`
Connected speaker avatars	Supabase `speakers` table

API needed: New query — `GET /api/beliefs/:id/connections` — returns connected speakers + agreement scores. Or compute client-side via DuckDB-WASM.

Interaction Spec

Action	Behavior
Tap ontology node	Navigates to a filtered view of all beliefs with that `worldview` or `core_axiom`
Tap connected speaker mini-card	Opens `<SpeakerProfileCard />` for that speaker
Tap connection line dot	Highlights the shared belief between speakers
Close (× button or swipe down on mobile)	Collapses back to `<BeliefCardFace />` (300ms reverse zoom)
Swipe left/right (mobile)	Navigate to prev/next belief in results

Acceptance Criteria

1. **Given** a belief with `core_axiom: "Sound money is essential"` and `worldview: "Austrian economics"`, **When** the expanded view renders, **Then** the ontology tree shows three nodes top-to-bottom: "Sound money is essential" → "Austrian economics" → atomic_belief text.
2. **Given** the expanded view, **When** connected speakers load, **Then** at least 1 and at most 5 connected speakers appear with visible connection lines to the ontology tree.
3. **Given** a 390px viewport, **When** the expanded view renders, **Then** the ontology tree appears below the card and connections appear below the tree, all scrollable vertically.
4. **Given** the user taps the close button, **Then** the view collapses back to the card grid within 300ms.
5. **Given** a connected speaker is tapped, **Then** the `<SpeakerProfileCard />` opens within 300ms.

Dependencies

- `<BeliefCardFace />` (embedded)
- Connected speakers query (DuckDB-WASM or new API endpoint)

- `<PixelAvatar size="sm" />`
- Speaker similarity data (`core_aggregations.parquet`)

Gap Analysis

Feature	v1 Status	v2 Action
Expanded belief view	Not implemented — cards are flat in v1	**Build from scratch**
Ontology tree	Data exists in parquet (`core_axiom` , `worldview`) but not visualized	**New** visualization
Connected speakers	`src/components/graph/speaker-graph.tsx` shows force graph but not per-belief	**New** per-belief connections query
Speaker similarity	`core_aggregations.parquet` has similarity matrix	**Reuse** data, build new UI

4.7 Belief Card — Social Embed

Screenshot Reference

Social Embed

Component Name

`<BeliefCardSocialEmbed />`

Description

A pre-rendered image card optimized for sharing on Twitter/X. Shows the @Bitcoinology branding, Bitcoin logo, the quote, speaker + episode reference, worldview badge, and app logo. When shared, tapping it deep-links to the belief in the app. This is a server-rendered OG image, not an interactive component.

Visual Requirements

Property	Value
Size	1200×630px (Twitter/X large card)
Background	`#0A0E1A`
Border	3px solid `#F7931A` , rounded 16px, outer glow `0 0 20px #F7931A60`
Top section	Bitcoin logo (48px) + "Bitcoin" text + "@Bitcoinology" handle
Quote area	`#1A1F2E` rounded rect inset, white text, 24px serif or `VT323`
Quote text	28px, `#FFFFFF` , max 3 lines, centered
Attribution	"Michael Saylor — WBD #412" in 18px `#9CA3AF`
Worldview badge	`<PixelBadge variant="orange" label="Worldview: Game Theory" />` style, bottom center

Property	Value
App logo	Bottom-right, 40px Bitcoin "₿" icon in orange circle
App name	"Bitcoinology" in 16px `#9CA3AF`, bottom center

Implementation: Next.js `@vercel/og` (Satori) or Puppeteer screenshot. Served from `GET /api/og/belief/:id`. Response is a PNG. OG meta tags reference this URL.

Data Requirements

Data	Source
`quote_text`	beliefs_summary.parquet (via API)
`speaker_name`	beliefs_summary.parquet
`episode_slug`	beliefs_summary.parquet
`worldview`	beliefs_summary.parquet
Belief ID for deep link	URL param

API endpoint: `GET /api/og/belief/:id` — returns PNG image. `GET /api/belief/:id` — returns HTML page with OG meta tags pointing to the image.

Interaction Spec

- **Not interactive in-app.** This is a generated image for social sharing.
- **Share flow:** User long-presses or taps share icon on `<BeliefCardFace />` → native share sheet or copy link → link generates OG card on paste.
- **Deep link:** Shared URL format: `https://bitcoinology.app/belief/:id` → opens app to that belief's expanded view.

Acceptance Criteria

1. **Given** a belief with ID "b_12345678", **When** `GET /api/og/belief/b_12345678` is called, **Then** a 1200×630px PNG is returned within 2 seconds.
2. **Given** the PNG is generated, **Then** the quote text fits within the card and is not truncated below 3 lines.
3. **Given** a tweet with the belief link, **When** Twitter renders the card, **Then** the large summary card displays with the correct image.
4. **Given** a user taps the shared link, **When** the app loads, **Then** the `<BeliefCardExpanded />` for that belief is displayed.

Dependencies

- `@vercel/og` or equivalent server-side image generation
- Belief data API endpoint
- Deep link routing in Next.js
- OG meta tag setup in page `<head>`

Gap Analysis

Does not exist in v1. No sharing, no OG images, no deep links. Entirely new.

4.8 Belief Card — Full Pixel Art

Screenshot Reference

Pixel Belief Card

Component Name

`<BeliefCardPixel />`

Description

Fully pixelated version of the belief card — avatar, text, border, confidence bar, and action icons are all rendered in an SNES RPG inventory card aesthetic. This is the in-app collectible format. Uses ornate pixel borders with corner decorations, a circular pixel avatar frame, and chunky pixel-font text.

Visual Requirements

Property	Value
Card size	300×420px, fixed aspect ratio
Border	8-bit ornate frame — use 9-slice sprite for corners + edges. Orange/brown palette from mockup.
Background	`#0A0E1A` (dark navy interior)
Avatar	Circular pixel frame, 64px, centered at top, `image-rendering: pixelated`
Quote text	`Press Start 2P` 12px (or `VT323` 18px), `#FFFFFF`, centered, max 5 lines
Speaker name	`Press Start 2P` 10px, `#F7931A`, left-aligned
Episode ref	`VT323` 14px, `#9CA3AF`, right-aligned on same line as name
Confidence bar	Full-width pixel health bar, `#F7931A` fill, segmented blocks, percentage right
Action icons row	5 pixel art icons (32×32px each), evenly spaced at bottom: Ontology, Worldview, Connections, Source, Share
Icon labels	`VT323` 8px, `#9CA3AF`, below each icon

9-slice border sprite: A single PNG (96×96px) with 32px corners and 32px edge tiles. CSS `border-image-slice: 32 fill` or rendered via canvas for pixel-perfect scaling.

Differentiator from `<BeliefCardFace />`: This card uses entirely pixel art assets. No CSS glow effects — all effects are baked into the border sprite. The avatar is always a generated pixel avatar, never a photo.

Data Requirements

Same as `<BeliefCardFace />` plus:

Data	Source
Pixel avatar	Generated per speaker per tier — stored in Supabase Storage or CDN

Interaction Spec

Same as `<BeliefCardFace />`. Tap → expand, icons → actions. Pixel art variant used when user toggles "Pixel Mode" or as default in certain views (e.g., collection/inventory view).

Acceptance Criteria

1. **Given** the pixel card renders, **Then** all images use `image-rendering: pixelated` and no anti-aliasing is visible on pixel art.
2. **Given** a speaker with no pixel avatar, **When** the card renders, **Then** a default pixel silhouette avatar is shown.
3. **Given** the 9-slice border, **When** the card is resized between 280–340px width, **Then** corners remain 32px and edges tile correctly without distortion.
4. **Given** the action icons, **Then** each icon has a 44×44px minimum touch target despite the 32px visual size.

Dependencies

- 9-slice border sprite (asset)
- Pixel avatar pipeline (see §4.14)
- `<PixelHealthBar />`
- `<PixelIcon />` set (pixel art variants)
- `Press Start 2P` font loaded

Gap Analysis

Does not exist in v1. Entirely new pixel art variant.

4.9 Community Belief Card

Screenshot Reference

Community Card

Component Name

`<CommunityBeliefCard />`

Description

User-created belief card with teal/cyan border to distinguish from speaker-sourced beliefs. Shows user's pixel avatar, their belief statement, research evidence (articles, clips, data) as pixel item icons, and a voting system with lightning bolt agrees (⚡) and sword challenges (⚔️). Community cards are CRAFTED by users; speaker cards are MINED from podcast data.

Visual Requirements

Property	Value
Card size	300×420px (same as pixel belief card)
Border	Teal/cyan (`#00E5CC`), pixel-art frame. Pixel variant: 9-slice with teal corners. Non-pixel variant: `2px solid #00E5CC`
Background	`#0A0E1A` with subtle teal inner glow
"COMMUNITY" header	`Press Start 2P` 12px, `#00E5CC`, centered at top, badge/banner style
User avatar	Basic pixel avatar, 48px, centered below header
Belief text	`VT323` 18px (or `Press Start 2P` 12px for pixel variant), `#FFFFFF`, centered, max 4 lines
Research section	Label "Research" in `VT323` 14px, then 2-3 pixel item icons (scroll, microphone, chart) representing evidence types
Topic tags	Teal `<PixelBadge variant="teal" />` — e.g., "Lightning Network", "Adoption"
Vote section	Two columns: Left = "⚡ 23 agrees", Right = "⚔ 4 challenges". `VT323` 16px, `#F7931A` for lightning, `#EF4444` for sword
"VOTE" label	`Press Start 2P` 10px, `#9CA3AF`, bottom
Devote count	Heart icon + count (e.g., "❤️ 20") for reputation/endorsements

Data Requirements

Data	Source	Notes
User ID + avatar	Supabase `users` table	Auth context
Belief text	Supabase `community_beliefs` table	**New table needed**
Research links	Supabase `community_belief_evidence` table	**New table needed**
Topic tags	Supabase `community_belief_tags` table	**New table needed**
Agree count	Supabase `community_votes` (type='agree')	**New table + RPC**
Challenge count	Supabase `community_votes` (type='challenge')	**New table + RPC**
Devote count	Supabase `community_devotes`	**New table**

New Supabase tables needed:

```
 community_beliefs (
  id uuid PRIMARY KEY,
  user_id uuid REFERENCES auth.users,
  belief_text text NOT NULL,
  created_at timestamptz DEFAULT now()
);

community_belief_tags (
  belief_id uuid REFERENCES community_beliefs,
  tag text NOT NULL
);

community_belief_evidence (
  id uuid PRIMARY KEY,
  belief_id uuid REFERENCES community_beliefs,
  evidence_type text CHECK (evidence_type IN ('article', 'podcast', 'data', 'video')),
  url text,
  title text
);

community_votes (
  id uuid PRIMARY KEY,
  belief_id uuid REFERENCES community_beliefs,
  user_id uuid REFERENCES auth.users,
  vote_type text CHECK (vote_type IN ('agree', 'challenge')),
  created_at timestamptz DEFAULT now(),
  UNIQUE (belief_id, user_id)
);
```

Interaction Spec

Action	Behavior
Tap card	Expand to show full evidence list + discussion thread
Tap ⚡ agrees	Cast agree vote (optimistic UI: count +1 immediately). Toggle if already voted.
Tap ⚔️ challenges	Cast challenge vote. Mutually exclusive with agree.
Tap research item	Open evidence link (external or in-app)
Tap topic tag	Filter community cards by that tag
Long press	Share menu (same as speaker cards)

Acceptance Criteria

1. **Given** a community belief card, **When** rendered, **Then** the border color is `#00E5CC` (teal) and visually distinct from orange speaker cards.
2. **Given** the user taps ⚡ , **When** the vote is cast, **Then** the count increments optimistically within 100ms and persists to Supabase within 2 seconds.
3. **Given** the user has already voted "agree", **When** they tap ⚔️ , **Then** the agree vote is removed and challenge vote is added (mutually exclusive).
4. **Given** a community card with 3 evidence items, **When** rendered, **Then** 3 pixel item icons are visible in the Research section.
5. **Given** the pixel variant, **Then** all elements use pixel art rendering and `image-rendering: pixelated`.

Dependencies

- New Supabase tables (schema above)
- Supabase RPC for vote counts
- `<PixelBorder variant="teal" />`
- `<PixelBadge variant="teal" />`
- `<PixelAvatar />` (basic user variant)
- `<PixelIcon />` (scroll, microphone, chart, sword, lightning)
- Auth context (who's voting)

Gap Analysis

Does not exist in v1. Community features are entirely new. v1 has no user-generated content, no voting, no community beliefs. All backend tables need to be created.

Speaker System

4.10 Speaker Profile Card

Screenshot Reference

Speaker Profile Card

Component Name

`<SpeakerProfileCard />`

Description

RPG character select screen for a speaker. Shows speaker photo with orange ring border, name + title, key stats (Episodes, Beliefs, Connections), belief breakdown by category as mini progress bars, and mini belief card thumbnails at the bottom.

Visual Requirements

Property	Value
Card size	Full-width on mobile (max 400px), 400×600px on larger screens
Background	`#0A0E1A`
Border	`2px solid #F7931A`, pixel art corner ornaments
Avatar	96px circle with 4px `#F7931A` ring, speaker photo (not pixel art)
Name	`Press Start 2P` 18px, `#FFFFFF`
Title	`VT323` 16px, `#9CA3AF` (e.g., "CEO, MicroStrategy")
Stats row	Three stats in a row: `#F7931A` large number (`Press Start 2P` 20px) + white label (`VT323` 12px)

Property	Value
Stat values	Episodes: count, Beliefs: count, Connections: count
Belief breakdown	4 horizontal bars showing topic distribution (e.g., "Sound Money 34%", "Game Theory 28%")
Breakdown bars	`<PixelHealthBar />` with category colors — orange for primary, dimmer for secondary
Mini belief cards	4 small card thumbnails (80×100px each) in a horizontal row at bottom, scrollable
Decorative	Pixel art zigzag border pattern along edges (visible in mockup)

Data Requirements

Data	Source
Speaker info (name, title, avatar)	Supabase `speakers` table
Episode count	DuckDB-WASM: `SELECT COUNT(DISTINCT episode_slug) FROM beliefs_summary WHERE speaker_slug = ?`
Belief count	DuckDB-WASM: `SELECT COUNT(*) FROM beliefs_summary WHERE speaker_slug = ?`
Connection count	`core_aggregations.parquet` → `speaker_similarity` matrix, count speakers with similarity > 0.3
Belief breakdown by topic	DuckDB-WASM: `SELECT topic, COUNT(*) FROM beliefs_summary WHERE speaker_slug = ? GROUP BY topic ORDER BY count DESC LIMIT 4`
Word count	Supabase `speakers` table or computed from `beliefs_summary`
Top beliefs (mini cards)	DuckDB-WASM: top 4 beliefs by `tier` DESC for this speaker

API: All computable client-side via DuckDB-WASM on `beliefs_summary.parquet` + `core_aggregations.parquet`. No new API needed.

Interaction Spec

Action	Behavior
Tap card	Opens `<SpeakerProfileExpanded />` (300ms panel slide)
Tap mini belief card	Opens that `<BeliefCardExpanded />`
Tap belief category bar	Filters to show only beliefs in that category
Swipe left/right on mini cards row	Scrolls through speaker's top beliefs

Acceptance Criteria

1. **Given** a speaker with 47 episodes, **When** the card renders, **Then** the "47" stat is displayed in `Press Start 2P` 20px orange.
2. **Given** a speaker with belief distribution [Sound Money: 34%, Game Theory: 28%, Technology: 22%, Philosophy: 16%], **When** the breakdown renders, **Then** 4 horizontal bars are shown with correct percentages and labels.

3. **Given** 4 mini belief cards, **When** the row renders, **Then** cards are horizontally scrollable on touch devices.
4. **Given** the card is tapped, **Then** `<SpeakerProfileExpanded />` slides in within 300ms.

Dependencies

- `<PixelAvatar size="lg" />` (with photo, not pixel art)
- `<PixelHealthBar />`
- `<BeliefCardFace />` (mini variant)
- `<PixelBorder variant="orange" />`
- DuckDB-WASM with `beliefs_summary.parquet` + `core_aggregations.parquet`
- Supabase `speakers` table

Gap Analysis

Feature	v1 Status	v2 Action
Speaker panel	`src/components/graph/speaker-panel.tsx` — basic info panel	**Redesign** as RPG character card
Speaker stats	Computed in `use-graph-data.ts`	**Reuse** computation, new UI
Speaker graph data	`src/app/api/graph/data/route.ts`	**Reuse** API, add topic breakdown
Mini belief cards	Not in v1	**New**

4.11 Speaker Profile Expanded

Screenshot Reference

Speaker Expanded

Component Name

`<SpeakerProfileExpanded />`

Description

Full view of a speaker profile showing their belief network as a force-directed graph (speaker node centered, belief clusters radiating outward). Sidebar shows top connections with other speakers and agreement scores. Every node in the graph is tappable.

Visual Requirements

Property	Value
Layout	Mobile: graph top (60vh), connections list below. Tablet+: graph left (60%), connections sidebar right (40%)
Background	`#000000` (pure black for graph contrast)

Property	Value
Header	Speaker avatar (64px) + name (`Press Start 2P` 16px) + stats row, top bar
Graph	`react-force-graph-2d` — orange nodes, orange link lines
Center node	Speaker avatar, 40px, orange ring, fixed position
Belief cluster nodes	12px circles, `#F7931A` , grouped by topic. Topic label on cluster centroid.
Link lines	`#F7931A40` (40% opacity), 1px
Topic labels	`VT323` 12px, `#F7931A` , positioned at cluster centers
Connections sidebar	List of speakers with avatars (32px), name, agreement % bar
Agreement bar	`<PixelHealthBar />` showing 0–100% agreement score
"TOP CONNECTIONS" header	`Press Start 2P` 12px, `#FFFFFF`
Speaker list items	Avatar + Name + Agreement score, 60px tall, tap target full width
Bottom row	Horizontally scrollable connected speaker cards (compact `<SpeakerProfileCard />` mini variant)

Data Requirements

Data	Source
Speaker node + all their beliefs	DuckDB-WASM: all beliefs for this speaker
Belief positions (for force graph)	Computed from 10-dim `weights` vector — project to 2D via t-SNE or UMAP (or use first 2 dims as x/y)
Topic clusters	Group beliefs by `topic` field
Connected speakers	`core_aggregations.parquet` → `speaker_similarity` matrix — top 10 most similar speakers
Agreement scores	Cosine similarity from `speaker_similarity` matrix
Connected speaker metadata	Supabase `speakers` table

Graph computation: Use `weights[0..1]` as initial x/y positions for the force graph, with `react-force-graph-2d` applying force simulation. Beliefs with the same `topic` are attracted to each other (custom force). Speaker connections are shown as edges between the center speaker node and other speaker nodes at the periphery.

Interaction Spec

Action	Behavior
Tap belief node in graph	Opens `<BeliefCardFace />` as popover near the tapped node
Tap topic cluster label	Filters sidebar to show only speakers connected via that topic

Action	Behavior
Tap connected speaker in sidebar	Navigates to that speaker's `<SpeakerProfileCard />`
Pinch zoom (mobile)	Zooms the force graph
Drag (desktop)	Pans the force graph
Tap center speaker node	Toggles between "by topic" and "by polarity" graph coloring

Acceptance Criteria

1. **Given** a speaker with 312 beliefs, **When** the graph renders, **Then** beliefs are clustered by `topic` with visible separation between clusters.
2. **Given** the force graph, **When** a belief node is tapped, **Then** a `<BeliefCardFace />` popover appears within 200ms near the tapped position.
3. **Given** the connections sidebar, **When** rendered, **Then** speakers are sorted by agreement score descending.
4. **Given** a 390px viewport, **Then** the graph takes 60% of viewport height and connections list is scrollable below.
5. **Given** the speaker has 10+ connections, **Then** only the top 10 are shown with a "Show all" link.

Dependencies

- `react-force-graph-2d`
- `<SpeakerProfileCard />` (entry point)
- `<BeliefCardFace />` (popover)
- `<PixelHealthBar />` (agreement bars)
- `<PixelAvatar />`
- DuckDB-WASM with both parquet files
- Force graph layout computation

Gap Analysis

Feature	v1 Status	v2 Action
Force graph	`src/components/graph/speaker-graph.tsx` — exists with `react-force-graph-2d`	**Enhance** — add topic clustering, belief node tapping, pixel art styling
Speaker panel	`src/components/graph/speaker-panel.tsx`	**Redesign** as RPG sidebar
Graph data API	`src/app/api/graph/data/route.ts`	**Reuse**, may need to add per-speaker filtering
Graph controls	`src/components/graph/graph-controls.tsx`	**Reuse/enhance**
Graph data hook	`src/hooks/use-graph-data.ts`	**Reuse/extend**

4.12 Speaker Profile Full Pixel RPG Sheet

Screenshot Reference

Pixel RPG Sheet

Component Name

`<SpeakerProfilePixel />`

Description

Full pixel RPG character sheet. Legendary portrait with laser eyes and ornate pixel frame at top, name in pixel font, stat block (Episodes, Beliefs, Connections, Words), skill bars by topic, and belief card thumbnails at bottom. This IS an RPG character select screen — the crowning visual element of the app.

Visual Requirements

Property	Value
Layout	Single column, scrollable, max-width 400px centered
Background	`#000000` or `#0A0E1A` with subtle pixel noise texture
Portrait frame	Ornate pixel border — gold for Legendary, purple for Epic, blue for Rare, gray for Common. Frame size 200×200px. Inner image is the AI-generated pixel avatar.
"LEGENDARY" banner	`Press Start 2P` 10px, gold text on dark banner ribbon below portrait
Name	`Press Start 2P` 20px, `#FFFFFF`, centered below banner
Stats section	Header "EPISODES" in `Press Start 2P` 14px. Then 4 rows: stat name (`VT323` 16px left) + value (`Press Start 2P` 16px right, `#F7931A`). Values: Episodes, Beliefs, Connections, Words.
Stats divider	Pixel art horizontal rule (1px dashed pattern)
Skills section	Header "SKILLS" in `Press Start 2P` 14px. Per-topic `<PixelHealthBar />` with topic name label. Top 4 topics.
Skill bar colors	Primary topic: `#F7931A` , secondary: `#C47615` , tertiary: `#8B6914`
Belief thumbnails	4 mini pixel belief cards at bottom (80×100px), in a row, each with tiny pixel art
Card border	Full-height pixel art frame wrapping the entire component

Data Requirements

Same as `<SpeakerProfileCard />` plus:

Data	Source
Pixel avatar URL (per tier)	Supabase Storage — `avatars/{speaker_slug}/{tier}.png`
Tier level	Computed from episode count (see §4.13)
Word count	Supabase `speakers` table → `total_words` column, or computed

Data	Source
Top 4 topics with belief counts	DuckDB-WASM
Top 4 beliefs (for thumbnails)	DuckDB-WASM, by `tier` DESC

Interaction Spec

Action	Behavior
Tap portrait	Opens full-screen pixel avatar view (pinch-to-zoom)
Tap stat row	For "Beliefs": shows belief list. For "Connections": shows connections graph.
Tap skill bar	Filters to beliefs in that topic
Tap mini belief card	Opens `<BeliefCardPixel />` expanded
Scroll	Full page scrollable

Acceptance Criteria

1. **Given** a Legendary speaker (50+ episodes), **When** the RPG sheet renders, **Then** the portrait frame is gold with "LEGENDARY" banner.
2. **Given** the speaker has 847K words, **When** the stats render, **Then** "Words" shows "847K" formatted.
3. **Given** the pixel avatar image, **Then** it renders at native pixel resolution with `image-rendering: pixelated`, no smoothing.
4. **Given** 4 skill topics, **When** bars render, **Then** bar widths are proportional to belief counts and total to 100%.
5. **Given** a Common tier speaker, **Then** the portrait frame is gray with no banner.

Dependencies

- Pixel avatar pipeline (§4.14, §6.4)
- `<PixelAvatar size="xl" />`
- `<PixelHealthBar />`
- `<PixelBorder variant="gold|purple|blue|gray" />`
- `<BeliefCardPixel />` (mini variant)
- Tier computation logic

Gap Analysis

Does not exist in v1. Entirely new. v1's `speaker-panel.tsx` shows basic info only.

4.13 Speaker Tier System

Screenshot Reference

Tier System

Component Name

`<SpeakerTierBadge />` (display), `computeSpeakerTier()` (logic)

Description

Speakers earn their tier (Common → Rare → Epic → Legendary) based on content contribution. Each tier unlocks a progressively more elaborate pixel avatar and visual effects. Tiers determine border color, glow effects, and portrait complexity across all speaker-related components.

Tier Definitions

Tier	Episodes	Words	Border Color	Glow	Avatar Style	Frame
🟫 Common	1–5	<10K	`#6B7280` (gray)	None	Simple 8-bit pixel portrait, muted colors	Plain gray rect
🟦 Rare	6–20	10K–50K	`#3498DB` (blue)	`0 0 12px #3498DB60`	Detailed pixel art, sharper features	Blue-highlighted frame
🟪 Epic	21–50	50K–100K	`#9B59B6` (purple)	`0 0 16px #9B59B680`	Full transformation: armor, energy effects	Ornate purple frame with decorations
🟧 Legendary	50+	100K+	`#FFD700` (gold)	`0 0 20px #FFD70060`	Laser eyes, lightning, boss-mode portrait	Gold ornate frame with animated effects

Tier computation logic:

```
function computeSpeakerTier(episodeCount: number, wordCount: number): 'common' | 'rare' | 'epic' | 'legen
  if (episodeCount >= 50 || wordCount >= 100_000) return 'legendary';
  if (episodeCount >= 21 || wordCount >= 50_000) return 'epic';
  if (episodeCount >= 6 || wordCount >= 10_000) return 'rare';
  return 'common';
}
```

Visual Requirements for `<SpeakerTierBadge />`

Property	Value
Size	80×24px
Background	Tier color at 20% opacity
Text	Tier name in `Press Start 2P` 8px, tier color
Border	1px solid tier color
Placement	Below speaker avatar or name in any speaker card

Data Requirements

Data	Source
Episode count per speaker	DuckDB-WASM: `COUNT(DISTINCT episode_slug)`
Word count per speaker	Supabase `speakers.total_words` or DuckDB-WASM aggregate of quote lengths

Acceptance Criteria

1. **Given** a speaker with 47 episodes and 847K words, **When** `computeSpeakerTier()` runs, **Then** it returns `'legendary'`.
2. **Given** a speaker with 3 episodes and 5K words, **When** `computeSpeakerTier()` runs, **Then** it returns `'common'`.
3. **Given** a speaker with 25 episodes and 8K words, **When** `computeSpeakerTier()` runs, **Then** it returns `'epic'` (episodes threshold met even though words is low).
4. **Given** a Legendary tier badge, **Then** the badge background is `#FFD70033`, text is `#FFD700`, border is `1px solid #FFD700`.

Dependencies

- Episode + word count data
- `<PixelBadge />` (extended for tiers)
- Tier colors defined in design tokens

Gap Analysis

Does not exist in v1. No tier system. Speaker data exists in `beliefs_summary.parquet` but no tier computation or visual differentiation.

4.14 Speaker Avatar Generation

Screenshot Reference

Legendary Avatar

Component Name

`<SpeakerAvatar />` (display), avatar generation pipeline (backend)

Description

Each speaker gets a unique AI-generated pixel avatar that evolves with their tier. Common tier gets a simple 8-bit portrait, while Legendary gets laser eyes, lightning effects, and boss-mode styling. Avatars are generated once per tier change and cached permanently.

Avatar Specifications per Tier

Tier	Resolution	Style Prompt Keywords	Effects
Common	128×128px	"simple 8-bit pixel portrait, muted colors, gray background, retro NES style"	None
Rare	128×128px	"detailed 16-bit pixel art portrait, blue glow, SNES style, sharper features"	Blue background glow
Epic	256×256px	"epic pixel art warrior portrait, armor, purple energy, ornate, boss character"	Purple energy crackling
Legendary	256×256px	"legendary pixel art portrait, laser eyes, lightning, Bitcoin symbol, boss mode, golden aura"	Laser eye beams, lightning bolts

Generation Pipeline

1. Speaker reaches new tier threshold
2. Backend job triggers AI image generation (DALL-E 3 or Stable Diffusion with pixel art LoRA)
3. Image is post-processed (ensure pixel-perfect scaling, remove anti-aliasing artifacts)
4. Stored in Supabase Storage: `avatars/{speaker_slug}/{tier}.png`
5. CDN URL is updated in `speakers` table

Visual Requirements

Property	Value
Rendering	Always use `image-rendering: pixelated` on `<img>`
Scaling	Common/Rare: 128px source, display at 64-128px. Epic/Legendary: 256px source, display at 64-256px.
Fallback	If no avatar exists, show a generic pixel silhouette with speaker's initial
Frame	Determined by tier — use `<PixelAvatar tier={tier} />`
Loading	Shimmer skeleton (pixel-art style) while avatar loads

Data Requirements

Data	Source
Avatar URL per tier	Supabase `speakers.avatar_urls` (JSONB: `{common: url, rare: url, ...}`)
Current tier	Computed from `computeSpeakerTier()`
Speaker slug	For storage path

Acceptance Criteria

1. **Given** a speaker at Legendary tier, **When** their avatar displays, **Then** the Legendary-tier image loads from `avatars/{slug}/legendary.png` with `image-rendering: pixelated`.
2. **Given** a speaker who just reached Rare tier, **When** the tier change is detected, **Then** a generation job is queued and the avatar updates within 5 minutes.
3. **Given** no avatar exists for a speaker, **Then** a pixel silhouette with their first initial is shown.

4. **Given** a slow network, **When** the avatar is loading, **Then** a pixel shimmer skeleton (64px) is displayed.

Dependencies

- AI image generation service (DALL-E 3 / Stable Diffusion API)
- Supabase Storage bucket `avatars`
- `speakers` table schema update (add `avatar_urls` JSONB column)
- Tier computation
- Background job runner (Motia step or cron)

Gap Analysis

Does not exist in v1. v1 has no avatars, no image generation, no tier-based visuals.

World Building

4.15 Event / Arena View

Screenshot Reference

No mockup exists yet. Conceptual design from the entity model.

Component Name

`<EventArenaView />`

Description

Events (podcast episodes, conferences, debates) displayed as "arenas" — visual stages where beliefs were spoken. Think boxing ring, colosseum, or stage set. Users browse events to see which beliefs were captured and which speakers appeared. Events are the SOURCE of belief cards.

Visual Requirements (Conceptual)

Property	Value
Layout	Full-pane view, header + scrollable content
Header	Event name in `Press Start 2P` 16px, date, podcast/venue name
Arena visual	Pixel art background depicting a stage/ring/arena. 390×200px hero image.
Speaker lineup	Horizontal row of `<PixelAvatar size="md" />` for speakers who appeared
Belief cards	Grid of `<BeliefCardFace />` extracted from this episode
Stats bar	"12 Beliefs Extracted • 2 Speakers • 45 min" in `VT323` 14px
Background	Dark with pixel art decorative border, arena-themed

Property	Value
Arena types	Podcast episode = boxing ring (orange), Conference = colosseum (gold), Debate = sword arena (red)

Data Requirements

Data	Source
Episode metadata	Supabase or HuggingFace dataset metadata
Episode slug	beliefs_summary.parquet → `episode_slug`
Podcast slug	beliefs_summary.parquet → `podcast_slug`
Speakers in episode	DuckDB-WASM: `SELECT DISTINCT speaker_slug, speaker_name FROM beliefs_summary WHERE episode_slug = ?`
Beliefs from episode	DuckDB-WASM: `SELECT * FROM beliefs_summary WHERE episode_slug = ? ORDER BY timestamp_start`
Arena type	Derived from podcast metadata or manual tagging

Interaction Spec

Action	Behavior
Tap speaker avatar	Opens `<SpeakerProfileCard />`
Tap belief card	Opens `<BeliefCardExpanded />`
Sort beliefs	By timestamp (chronological) or by confidence
Filter by speaker	Tap speaker avatar to filter beliefs to that speaker only

Acceptance Criteria

1. **Given** an episode with 2 speakers, **When** the arena renders, **Then** both speaker avatars are visible in the lineup.
2. **Given** 12 beliefs extracted from an episode, **When** the grid renders, **Then** all 12 cards are accessible (scrollable).
3. **Given** the user taps a speaker in the lineup, **Then** the belief grid filters to that speaker's beliefs only.
4. **Given** the event view, **Then** "12 Beliefs Extracted • 2 Speakers" stats are accurate from the data.

Dependencies

- Episode metadata (may need new Supabase table or extend existing)
- `<BeliefCardFace />`
- `<PixelAvatar />`
- DuckDB-WASM
- Arena background pixel art assets (3 variants)

Gap Analysis

Does not exist in v1. v1 has no event/episode browsing view. Episode data exists in parquet (`episode_slug` , `podcast_slug`) but no dedicated UI.

4.16 Organization / Guild View

Screenshot Reference

No mockup exists yet. Conceptual design from the entity model.

Component Name

`<OrganizationGuildView />`

Description

Organizations (MicroStrategy, Swan Bitcoin, podcast networks) displayed as "guilds" or "factions." Users browse guild members, collective beliefs, and faction alignment. Guilds show how organizations cluster around shared worldviews.

Visual Requirements (Conceptual)

Property	Value
Layout	Full-pane view, guild banner + members + beliefs
Guild banner	Pixel art banner/crest, organization name in `Press Start 2P` 18px
Guild color	Derived from organization's primary brand color, applied as accent
Members section	Grid of `<SpeakerProfileCard />` (compact) for speakers associated with this org
Collective beliefs	"Guild Beliefs" — aggregated top beliefs across all members
Faction alignment	Radar chart or bar chart showing guild's 10-dim weight centroid
Stats	"5 Members • 847 Beliefs • Active Since 2020"
Allegiance indicator	"Allied with: [Guild X]" — guilds with similar centroids

Data Requirements

Data	Source	Notes
Organization metadata	**New** Supabase table `organizations`	Name, description, logo, brand color
Speaker-org mapping	**New** Supabase table `speaker_organizations`	Many-to-many
Guild centroid	Computed: average of all member speakers' belief weight centroids	From `core_aggregations.parquet`
Collective top beliefs	DuckDB-WASM: top beliefs across all member speakers	By `tier` DESC
Allied guilds	Computed: orgs with highest centroid cosine similarity	From centroids

New Supabase tables:

```
organizations (
  id uuid PRIMARY KEY,
  name text NOT NULL,
  slug text UNIQUE NOT NULL,
  description text,
  logo_url text,
  brand_color text, -- hex color
  created_at timestamptz DEFAULT now()
);

speaker_organizations (
  speaker_slug text REFERENCES speakers(slug),
  organization_id uuid REFERENCES organizations(id),
  role text, -- e.g., "CEO", "Host", "Contributor"
  PRIMARY KEY (speaker_slug, organization_id)
);
```

Interaction Spec

Action	Behavior
Tap member	Opens `<SpeakerProfileCard />`
Tap collective belief	Opens `<BeliefCardExpanded />`
Tap allied guild	Navigates to that guild's view
Filter by role	Show only "Hosts" or "Contributors"

Acceptance Criteria

1. **Given** an organization with 5 speakers, **When** the guild view renders, **Then** all 5 speaker cards are displayed.
2. **Given** the guild's centroid weights, **When** the alignment chart renders, **Then** all 10 dimensions are represented.
3. **Given** guild alliances are computed, **Then** the top 3 most similar guilds are shown.
4. **Given** the user navigates to a guild, **Then** the page loads within 2 seconds.

Dependencies

- New Supabase tables (`organizations`, `speaker_organizations`)
- Manual data entry for org-speaker mappings
- Centroid computation logic
- `<SpeakerProfileCard />` (compact variant)
- `<BeliefCardFace />`
- Radar chart component (new, or use simple bar chart)

Gap Analysis

Does not exist in v1. No organization concept in v1. Requires new database tables and manual data population.

Part 5: Agent Architecture

> For the complete agent blueprint, see `specs/AGENT_BLUEPRINT.md` (if available) and `agentic-workflow-flow.md`.

5.1 The Three Agents

🔮 Oracle (Search Agent)

- **Trigger:** User types a search query
- **Job:** Fast belief retrieval + synthesis
- **Flow:** Query → Embed → Vector Search (Qdrant) → Rank → Synthesize → Response
- **Speed:** 2-5 seconds
- **Workflow tree:** 3 nodes (SEARCH → RANK → SYNTHESIZE)

🐺 Jackal (Deep Research Agent)

- **Trigger:** User clicks "Go Deeper" or Oracle finds thin coverage
- **Job:** Multi-hop reasoning across beliefs, connecting dots
- **Flow:** Seed Query → Decompose → Parallel Sub-Queries → Cross-Reference Beliefs → Build Argument Graph → Narrative Synthesis
- **Speed:** 10-30 seconds (this is where the animation matters most)
- **Workflow tree:** Branching tree with parallel threads

📖 Playbook (Guided Exploration)

- **Trigger:** User selects a Playbook topic
- **Job:** Structured learning path through the belief graph
- **Flow:** Topic → Load Playbook Template → Sequence Beliefs → Present Step-by-Step → Branch on User Choice
- **Workflow tree:** Sequential with branching at choice points

5.2 SSE Event Stream Spec

Each workflow step emits events the frontend consumes via Server-Sent Events:

```
{"type": "step_start", "id": "vector_search", "label": "🔍 Vector Search"}
{"type": "step_progress", "id": "vector_search", "data": {"found": 23}}
{"type": "step_complete", "id": "vector_search", "duration_ms": 340}
{"type": "step_start", "id": "ranking", "label": "⚖️ Ranking & Filtering"}
{"type": "stream_token", "id": "synthesis", "token": "Michael"}
{"type": "stream_token", "id": "synthesis", "token": " Saylor"}
{"type": "response.ready", "answer": "...", "citations": [...]}
```

Client library: `@motiadev/stream-client-react` (exists in v1)

5.3 Agent Coordination Model

```
┌─────────┐    ┌──────────┐    ┌───────────┐
│ Next.js │───▶│  Motia   │───▶│ AI Agents │
│Frontend │◀───│ (flows)  │◀───│ (Claude)  │
│         │ SSE│          │    └─────┬─────┘
└─────────┘    └────┬─────┘          │
                    │          ┌─────▼─────┐
                    │          │  Qdrant   │
                    │          │ (vectors) │
                    │          └───────────┘
                    │          ┌───────────┐
                    └─────────▶│ Supabase  │
                               │(metadata) │
                               └───────────┘
```

- Motia orchestrates all agent flows with discrete steps
- Steps emit real-time status via SSE
- Steps can run in parallel (vector search + metadata lookup)
- Failed steps show red in the tree (with retry option)
- Each step logs to Supabase for analytics

5.4 Workflow Visualization Mapping

Agent	Workflow Tree Shape	Nodes
Oracle (direct)	Linear (3 nodes)	SEARCH → RANK → SYNTHESIZE
Oracle (multi-perspective)	Linear (4 nodes)	QUICK TAKE → DEEP ANALYSIS → STRATEGIC VIEW → SYNTHESIZE
Playbook	Branching (4+ nodes)	SEARCH → ANALYZE (4 parallel lenses) → SYNTHESIZE
Jackal	Tree (N branches)	SEARCH → DECOMPOSE → PARALLEL SEARCH (N) → CROSS-REF → SYNTHESIZE

What makes this different: 1. **Transparency** — Users see exactly what's happening, not a black box 2. **Education** — The tree teaches users how belief extraction works 3. **Trust** — Every claim links back to a specific quote + episode + timestamp 4. **Engagement** — Watching the tree build is genuinely fun

Part 6: Technical Architecture

6.1 Stack Overview

Layer	Technology	Purpose
Frontend	Next.js 14 (App Router)	SSR, routing, React
UI Framework	Custom pixel art design system	Retro aesthetic components
State	Zustand	Client-side state management
Client DB	DuckDB-WASM + IndexedDB	Client-side parquet queries, offline cache
Vector Search	Qdrant	Semantic search over belief embeddings
Database	Supabase (PostgreSQL)	Auth, speakers, metadata, community
Orchestration	Motia	Agent workflow orchestration + SSE streaming
AI	Claude (Anthropic)	Synthesis, analysis, decomposition
Cache	Redis	API response caching, rate limiting
Image Gen	DALL-E 3 / Stable Diffusion	Pixel avatar generation
CDN	Supabase Storage + CDN	Avatar images, sprite assets

6.2 Data Model

Existing Tables (v1)

Table	Key Columns	Status
`speakers`	`slug`, `name`, `avatar_url`, `bio`	Exists — needs `avatar_urls` JSONB, `total_words`
`auth.users`	Supabase Auth	Exists

Existing Parquet Files (v1)

File	Key Fields	Status
`beliefs_summary.parquet`	`id`, `speaker_name`, `speaker_slug`, `atomic_belief`, `quote_text`, `polarity`, `polarity_confidence`, `core_axiom`, `worldview`, `topic`, `episode_slug`, `podcast_slug`, `timestamp_start`, `weights[10]`	Exists
`core_aggregations.parquet`	`speaker_similarity` matrix, cluster centroids	Exists

New Tables (v2)

```
-- Community Beliefs
community_beliefs (
  id uuid PRIMARY KEY,
  user_id uuid REFERENCES auth.users,
  belief_text text NOT NULL,
  created_at timestamptz DEFAULT now()
);
```

```
community_belief_tags (
  belief_id uuid REFERENCES community_beliefs,
  tag text NOT NULL
);

community_belief_evidence (
  id uuid PRIMARY KEY,
  belief_id uuid REFERENCES community_beliefs,
  evidence_type text CHECK (evidence_type IN ('article', 'podcast', 'data', 'video')),
  url text,
  title text
);

community_votes (
  id uuid PRIMARY KEY,
  belief_id uuid REFERENCES community_beliefs,
  user_id uuid REFERENCES auth.users,
  vote_type text CHECK (vote_type IN ('agree', 'challenge')),
  created_at timestamptz DEFAULT now(),
  UNIQUE (belief_id, user_id)
);

-- Organizations / Guilds
organizations (
  id uuid PRIMARY KEY,
  name text NOT NULL,
  slug text UNIQUE NOT NULL,
  description text,
  logo_url text,
  brand_color text,
  created_at timestamptz DEFAULT now()
);

speaker_organizations (
  speaker_slug text REFERENCES speakers(slug),
  organization_id uuid REFERENCES organizations(id),
  role text,
  PRIMARY KEY (speaker_slug, organization_id)
);
```

Schema Updates

Table	Change
`speakers`	Add `avatar_urls JSONB`, `total_words INTEGER`

6.3 API Endpoints

Existing (v1 — Reuse)

Endpoint	Method	Purpose
`/api/query` (`/chat`)	POST	Main search — triggers Motia flow, returns SSE stream
`/api/graph/data`	GET	Speaker graph data

New (v2)

Endpoint	Method	Purpose
`/api/og/belief/:id`	GET	Generate OG social card image (PNG)
`/api/belief/:id`	GET	Belief detail page with OG meta tags
`/api/beliefs/:id/connections`	GET	Connected speakers for a belief
`/api/community/beliefs`	GET/POST	List/create community beliefs
`/api/community/beliefs/:id/vote`	POST	Cast agree/challenge vote
`/api/community/beliefs/:id/evidence`	POST	Attach evidence to belief
`/api/speakers/:slug/avatar`	POST	Trigger avatar generation (admin)

6.4 Asset Pipeline

Speaker Pixel Avatar Generation

Trigger: Speaker tier changes (new episode processed → episode count crosses threshold).

Pipeline:

```
 1. Tier change detected (CocoIndex pipeline or manual trigger)
    └── Emit event: { speaker_slug, new_tier, episode_count, word_count }

2. Avatar generation job (Motia background step)
   ├── Fetch speaker photo from speakers.avatar_url (reference photo)
   ├── Build prompt from tier template + speaker description
   │   └── "Create a {tier_style} pixel art portrait of a {description}.
   │        Style: {style_keywords}. Background: {bg_keywords}."
   ├── Call AI image API (DALL-E 3 or Stable Diffusion XL with pixel art LoRA)
   │   └── Request: 1024×1024 PNG
   ├── Post-process:
   │   ├── Downscale to target resolution (128px or 256px) using nearest-neighbor
   │   ├── Remove anti-aliasing artifacts
   │   ├── Apply palette restriction (NES 54-color palette or custom)
   │   └── Validate pixel-art quality (no smooth gradients)
   └── Output: PNG file

3. Storage
   ├── Upload to Supabase Storage: avatars/{speaker_slug}/{tier}.png
   ├── Generate CDN URL
   └── Update speakers table: avatar_urls JSONB

4. Cache
   ├── CDN caching: immutable, 1 year max-age
   ├── Cache key: {speaker_slug}-{tier}-{version}
   └── Client: preload current tier avatar via <link rel="preload">
```

Cost estimate: - DALL-E 3: ~$0.04 per avatar (1024×1024) - ~100 speakers × 4 tiers = 400 avatars = ~$16 one-time - Incremental: ~$0.04 per new tier upgrade

Pixel Art Sprite Assets

Asset	Size	Count	Format
Runner sprite sheet	128×48px	1	PNG
Coach sprite sheet	128×48px	1	PNG
9-slice card border (orange)	96×96px	1	PNG
9-slice card border (teal)	96×96px	1	PNG
9-slice card border (gold/purple/blue/gray)	96×96px each	4	PNG
Pixel icons (16×16)	16×16px	12+	PNG sprite sheet (192×16)
Arena backgrounds	390×200px	3	PNG
Guild banner templates	300×80px	3	PNG
Default avatar silhouette	128×128px	1	PNG

Asset Storage Structure

```
Supabase Storage:
  avatars/
    michael-saylor/
      common.png, rare.png, epic.png, legendary.png
    lyn-alden/
      ...

/public/sprites/
  borders/      → orange-9slice.png, teal-9slice.png, gold/purple/blue/gray
  characters/   → runner-sheet.png, coach-sheet.png
  icons/        → pixel-icons-16.png, pixel-icons-32.png (sprite sheets)
  backgrounds/  → arena-boxing.png, arena-colosseum.png, arena-debate.png
  ui/           → default-avatar.png, health-bar-segment.png, checkmark-pixel.png
```

6.5 Performance Budget

Image Sizes

Asset Type	Max Size	Format
Pixel avatar (128px)	15 KB	PNG
Pixel avatar (256px)	40 KB	PNG
Sprite sheet	10 KB	PNG
9-slice border	5 KB	PNG
Arena background	50 KB	PNG/WebP
OG social card	150 KB	PNG
Total initial sprites	< 100 KB	PNG

Animation Frame Rates

Animation	Target FPS	Method
Sprite animations	6-8 FPS (150ms/frame)	CSS `steps()` — intentionally retro
Lightning bolt	60 FPS	SVG + `requestAnimationFrame`
Force graph	60 FPS	Canvas via `react-force-graph-2d`
Card transitions	60 FPS	CSS transforms + `will-change`
Glow pulse	60 FPS	CSS animation
Energy particle flow	30 FPS	Canvas or CSS

Lazy Loading Strategy

Resource	Strategy
Speaker pixel avatars	`loading="lazy"` on `<img>`, preload only visible cards
`react-force-graph-2d`	Dynamic import `next/dynamic` — only load on graph view
DuckDB-WASM	Initialize on first search, not on app load (~2MB WASM)
`beliefs_summary.parquet`	Fetch on first data query, cache in IndexedDB
`core_aggregations.parquet`	Fetch on first graph/profile view
Sprite sheets	Preload in `<head>` — small enough for eager load

Resource	Strategy
OG image generation	On-demand, cached at CDN edge

Bundle Size Targets

Chunk	Max Size (gzipped)
Initial JS bundle	< 150 KB
Force graph chunk	< 80 KB (dynamic)
DuckDB-WASM	~2 MB (deferred)
Total first paint resources	< 300 KB

Core Web Vitals Targets

Metric	Target
LCP	< 2.0s
INP	< 150ms
CLS	< 0.05
TTFB	< 500ms
FCP	< 1.5s

6.6 Accessibility

Pixel Art Accessibility

Requirement	Implementation
Alt text for avatars	`alt="{Speaker Name} pixel art avatar, {tier} tier"`
Alt text for belief cards	`alt="Belief card: {atomic_belief} — {speaker_name}"`
Alt text for animations	`role="img" aria-label="Retro search animation: searching for beliefs"`
Screen reader for health bars	`role="progressbar" aria-valuenow={94} aria-valuemin={0} aria-valuemax={100} aria-label="Confidence: 94%"`
Screen reader for tier system	`aria-label="Speaker tier: Legendary"`
Animation pause	Respect `prefers-reduced-motion`: disable sprite animations, lightning bolt, glow pulses. Show static states instead.

High Contrast Mode

When `prefers-contrast: high` is detected:

Change	Implementation
Remove glow effects	`box-shadow: none`
Increase border width	`2px → 3px`
Increase text contrast	Ensure all text meets WCAG AAA (7:1 ratio)
Remove background gradients	Solid `#000000` backgrounds
Orange on dark passes	`#F7931A` on `#0A0E1A` = 5.2:1 ✅ (AA) — for AAA, lighten to `#FFB347` on `#000000` = 10.4:1

Keyboard Navigation

Context	Keys
Results grid	Arrow keys navigate between cards, Enter opens expanded view
Expanded card	Tab cycles through ontology nodes → connected speakers → close button. Escape closes.
Workflow tree	Tab cycles through nodes, Enter expands node details
Force graph	Arrow keys pan, +/- zoom, Tab cycles through speaker nodes
Search bar	Auto-focus on page load, Enter submits

Color Blindness

Concern	Mitigation
Orange vs Red polarity	Add icon differentiation: ✓ for "for", × for "against", — for "neutral" in addition to color
Teal vs Blue tier	Tier badges include text label ("RARE", "COMMUNITY") not just color
Vote buttons	⚡ and ⚔️ icons provide meaning independent of color

---

Part 7: Implementation Roadmap

7.1 Phase 1: Foundation

Goal: Design system + shared components + search flow skeleton

What's included: - CSS custom properties (color tokens, typography, spacing) - All 7 shared components (`<PixelBorder />`, `<PixelHealthBar />`, `<PixelAvatar />`, `<PixelBadge />`, `<PixelIcon />`, `<PixelButton />`, `<RetroSearchBar />`) - `<SearchLoadingAnimation />` with sprite assets - `<LightningStrikeTransition />` SVG animation - `<LightningWorkflowTree />` connected to existing SSE stream - `<ResultsGrid />` basic layout (cards use placeholder styling)

Estimated effort: 2-3 weeks
Dependencies: Sprite sheet pixel art assets (can use placeholders initially)
What ships: Users get the animated search experience. Results still show, just with basic card styling.

7.2 Phase 2: Cards

Goal: Full belief card system + speaker profiles

What's included: - `<BeliefCardFace />` with confidence bar, polarity, action icons - `<BeliefCardExpanded />` with ontology tree + connections - `<BeliefCardPixel />` pixel art variant - `<SpeakerProfileCard />` RPG character card - `<SpeakerProfileExpanded />` force graph enhancement - `<SpeakerProfilePixel />` full RPG sheet - Tier computation logic (`computeSpeakerTier()`) - `<SpeakerTierBadge />`

Estimated effort: 3-4 weeks
Dependencies: Phase 1 shared components, DuckDB-WASM queries for connections
What ships: Full card interaction loop — search → card → expand → explore speaker → explore connections.

7.3 Phase 3: Agents

Goal: Full workflow visualization + agent integration

What's included: - Jackal parallel branching in workflow tree - Playbook multi-lens visualization - Oracle multi-perspective mode - Failed step retry UI - Node metadata expansion (tap completed node) - "Go Deeper" trigger for Jackal

Estimated effort: 2-3 weeks
Dependencies: Phase 1 workflow tree, Motia SSE event extensions
What ships: All three agent types have distinct, beautiful workflow visualizations.

7.4 Phase 4: Community

Goal: User-generated content + social sharing

What's included: - `<CommunityBeliefCard />` creation flow - Evidence attachment UI - Voting system (⚡ agrees / ⚔️ challenges) - `<BeliefCardSocialEmbed />` OG image generation - Deep link routing (`/belief/:id`) - New Supabase tables (community_beliefs, votes, evidence) - Share flow (native share sheet integration)

Estimated effort: 3-4 weeks
Dependencies: Phase 2 card system, Supabase auth, new database tables
What ships: Community participation loop — create, share, vote. Social sharing drives growth.

7.5 Phase 5: World Building

Goal: Events/arenas + organizations/guilds + avatar generation

What's included: - `<EventArenaView />` with pixel art arena backgrounds - `<OrganizationGuildView />` with faction alignment - Speaker avatar generation pipeline (DALL-E 3 integration) - New Supabase tables (organizations, speaker_organizations) - Manual org-speaker mapping data entry - Arena background pixel art assets (3 variants) - Guild banner assets

Estimated effort: 4-5 weeks
Dependencies: Phase 2 speaker system, Phase 4 community tables, AI image API access
What ships: The full world — arenas, guilds, evolving avatars. The game is complete.

Appendix

A. Gap Analysis Table

Component	v1 Status	v2 Action	Phase
Search Loading Animation	Generic spinner	**Build from scratch**	1
Lightning Strike Transition	Does not exist	**Build from scratch**	1
Lightning Workflow Tree	Does not exist	**Build from scratch**	1
Results Grid	Basic card list + chat format	**Redesign** with pixel grid	1
Belief Card — Face	Basic shadcn card, no pixel art	**Redesign** completely	2
Belief Card — Expanded	Does not exist	**Build from scratch**	2
Belief Card — Social Embed	Does not exist	**Build from scratch**	4
Belief Card — Full Pixel	Does not exist	**Build from scratch**	2
Community Belief Card	Does not exist (no community features)	**Build from scratch**	4
Speaker Profile Card	Basic info panel	**Redesign** as RPG card	2
Speaker Profile Expanded	Force graph exists	**Enhance** with clustering + pixel art	2
Speaker Profile Pixel RPG	Does not exist	**Build from scratch**	2
Speaker Tier System	Does not exist	**Build from scratch**	2
Speaker Avatar Generation	No avatars	**Build from scratch**	5
Event / Arena View	Does not exist	**Build from scratch**	5
Organization / Guild View	Does not exist	**Build from scratch**	5
SSE Streaming	Exists (`@motiadev/stream-client-react`)	**Reuse + extend**	1
DuckDB-WASM	Exists for parquet queries	**Reuse + extend**	1
Zustand State	Exists (`app-store.ts`)	**Extend**	1
Force Graph	Exists (`react-force-graph-2d`)	**Enhance**	2
Community DB Tables	Do not exist	**Create** (5 new tables)	4
Organization DB Tables	Do not exist	**Create** (2 new tables)	5

B. Glossary

Term	Definition
Belief Card	An atomic claim extracted from a podcast transcript. The core unit of content — collectible, shareable, inspectable.
Arena	The game metaphor for events (podcast episodes, conferences). Where beliefs are spoken.
Guild	The game metaphor for organizations (companies, networks). Groups of speakers with shared worldviews.
Tier	Speaker rarity level (Common → Rare → Epic → Legendary) based on contribution volume.
Oracle	Fast search agent — retrieves and synthesizes beliefs in 2-5 seconds.
Jackal	Deep research agent — multi-hop reasoning across beliefs, 10-30 seconds.
Playbook	Guided exploration agent — structured learning paths through the belief graph.
Ontology Tree	Hierarchical classification: Core Axiom → Worldview → Claim.
Polarity	Whether a belief is "for", "against", or "neutral" on its topic.
Confidence	How strongly a speaker holds a belief (0-100%), derived from language analysis.
Community Card	User-created belief (teal border) backed by research evidence and community votes.
Pixel Avatar	AI-generated pixel art portrait of a speaker, evolving with tier.
9-slice	A technique for scalable pixel art borders using a 96px source image with 32px corners.
SSE	Server-Sent Events — real-time unidirectional streaming from Motia to the frontend.
Workflow Tree	Visual pipeline showing agent processing steps in real-time (the lightning bolt).
DuckDB-WASM	Client-side SQL engine for querying parquet files directly in the browser.
Motia	Agent workflow orchestration framework that powers all AI flows.

C. Reference Links

Resource	Location
This PRD	`PRD_BITCOINOLOGY_V2.md` (repo root)
Design Requirements (full specs)	`specs/DESIGN_REQUIREMENTS.md`
Design Deck (visual narrative)	`designs/DESIGN_DECK.md`
Design Extraction Spec (v1→v2 port)	`specs/V2_DESIGN_EXTRACTION_SPEC.md`
Agentic Workflow Flow	`../../agentic-workflow-flow.md`
v1 Codebase	`../be-bitcoinology-v1/`
v1 Original PRD	`../be-bitcoinology-v1/_bmad-output/planning-artifacts/prd.md`

Resource	Location
Design Mockups — Retro UX	`designs/retro-ux/`
Design Mockups — Belief Cards	`designs/belief-cards/`
Design Mockups — Profile Cards	`designs/profile-cards/`
Design Mockups — Pixel Cards	`designs/pixel-cards/`

Built by Max Power ⚡ — Belief Engines
"The Matrix, but orange."