I want to add an additional perspective on the cards view. The existing screen and code should stay as is. I would like the image shown in the mockup to be presented from a button in a right hand vertical panel. Can you read the document an produce an implementation plan. I may be moving to a graph at some point.

Visual Goal Mockup.

📂 1. Configuration: `pathfinder_config.json`

This defines the "Mission Seeds." If a card is a descendant of these seeds in `relationships.json`, it will glow.

JSON

```
{
  "trails": [
    {
      "id": "legacy_rescue",
      "name": "Legacy Rescue (Operational Bedrock)",
      "color": "#3498db",
```

```
      "seeds": ["002_stack_data_engineering", "009_stack_real_world",
"006_stack_legacy_modernisation"],
      "description": "Expert core recovery and migration strategy."
    },
    {
      "id": "safe_innovation",
      "name": "Safe Innovation (Agentic Modernisation)",
      "color": "#2ecc71",
      "seeds": ["008_stack_ai_development", "1442_the_fruit_heading",
"005_stack_business_analysis"],
      "description": "Bridging AI awareness with engineering safety."
    },
    {
      "id": "systems_architect",
      "name": "Systems Architect (Enterprise Guardian)",
      "color": "#e74c3c",
      "seeds": ["011_stack_governance_compliance", "1417_the_taproot_heading",
"1421_the_trunk_heading"],
      "description": "Foundational logic and strict governance standards."
    }
  ]
}
```

🛠️ 2. Logical Engine: `trailLogic.js`

Optimized for a static site. This calculates the recursive path once and caches it to prevent browser lag.

JavaScript

```
/**
 * PATHFINDER TRAVERSAL ENGINE
 * Identifies all descendant nodes based on relationships.json
 */
const trailCache = {};

export const getNodesInTrail = (trailId, trailsConfig, relationships) => {
  if (trailCache[trailId]) return trailCache[trailId];

  const config = trailsConfig.find(t => t.id === trailId);
  if (!config) return [];

  let activeNodes = new Set(config.seeds);
```

```
  let stack = [...config.seeds];

  while (stack.length > 0) {
    const currentNodeId = stack.pop();
    // Filter relationships to find children
    const children = relationships
      .filter(rel => rel.from === currentNodeId && rel.type === "contains")
      .map(rel => rel.to);

    children.forEach(childId => {
      if (!activeNodes.has(childId)) {
        activeNodes.add(childId);
        stack.push(childId);
      }
    });
  }

  const resultArray = Array.from(activeNodes);
  trailCache[trailId] = resultArray;
  return resultArray;
};
```

💅 3. CSS Specifications: `trailStyles.css`

Visual logic for the "Nuts and Bolts" view.

CSS

```
/* Highlighting the active path */
.node-active-trail {
  transition: all 0.4s cubic-bezier(0.4, 0, 0.2, 1);
  z-index: 100 !important;
  pointer-events: auto;
}

/* Glow Effects per Trail */
.legacy_rescue-glow { filter: drop-shadow(0 0 10px rgba(52, 152, 219, 0.9)); border: 2px solid
#3498db !important; }
.safe_innovation-glow { filter: drop-shadow(0 0 10px rgba(46, 204, 113, 0.9)); border: 2px solid
#2ecc71 !important; }
.systems_architect-glow { filter: drop-shadow(0 0 10px rgba(231, 76, 60, 0.9)); border: 2px solid
#e74c3c !important; }
```

```
/* The "Blackout" Dimming logic */
.node-dimmed {
  opacity: 0.1;
  filter: grayscale(100%) blur(1px);
  pointer-events: none; /* User cannot interact with non-trail nodes */
  transition: all 0.4s ease;
}

/* Sidebar Button Styling (Nuts and Bolts Panel) */
.trail-btn {
  width: 100%;
  padding: 12px;
  margin-bottom: 8px;
  background: #1a1a1a;
  border: 1px solid #333;
  color: #ccc;
  cursor: pointer;
  text-align: left;
}
.trail-btn:hover { background: #252525; color: white; }
.trail-btn.active { border-left: 4px solid var(--trail-color); color: white; }
```

🤖 4. The Antigravity Master Prompt

Copy/Paste this entirely for your programmer.

> **"SYSTEM UPGRADE: STRATEGIC NAVIGATOR CORE"**
>
> **Task:** Implement client-side path filtering for our static Skillsweb application.
>
> **Requirements:**
>
> 1. **Data Integration:** Load `pathfinder_config.json` and initialize an `activeTrailId` state (default: null).
> 2. **Sidebar Controls:** Create three navigation buttons in the right-hand panel. On click, set `activeTrailId`. Add a 'Reset' button to clear state.
> 3. **Connectivity Logic:** Use the `getNodesInTrail` function (provided) to compute the set of visible IDs based on the `contains` relationships starting from the config seeds.
> 4. **DOM Updates:**
>   - If `activeTrailId` is NOT null:

        - Apply `.node-active-trail` and `[trailId]-glow` to all nodes in the computed set.
        - Apply `.node-dimmed` to all other nodes.
        - Update the breadcrumb text color to match the trail color.
    - If `activeTrailId` is null:
        - Remove all trail-related classes. Restore 100% opacity to all nodes.
5. **Performance:** Ensure the recursive traversal is memoized to prevent re-calculation on every UI render.

📋 The "Final Receipt" Validation

By dropping this in, your app will transition from a "Search Tool" to a **"Strategic Narrative Tool."**

1. **Immediate Clarity:** A recruiter doesn't have to guess. They click a mission.
2. **Visual Proof:** Your 30 years of technical depth (The orphans) will physically glow when they select the "Legacy Rescue" mission.
3. **Static Efficiency:** No backend, no latency—just pure logic running in the browser.