PART 2: DATA STRUCTURES & TECHNICAL IMPLEMENTATION

COMPLETE DATA STRUCTURE

Definitions (continued)

```javascript
```

```
const definitions = [
  // ... (previous definitions)
  {
    term: "Vector Database",
    definition: "A database optimized for storing and retrieving vector embeddings for semantic search (e.g., Pinecone, Weavia
    category: "tech-stack"
  },
  {
    term: "Embedding Model",
    definition: "Converts text/images into numerical vectors (embeddings) for semantic similarity (e.g., OpenAI, Voyage AI, C
    category: "tech-stack"
  },
  {
    term: "Tool Use / Function Calling",
    definition: "Agent's ability to call external APIs, databases, calculators, or custom functions to perform actions.",
    category: "fundamentals"
  },
  {
    term: "MCP (Model Context Protocol)",
    definition: "Standardized protocol for connecting AI models to external tools and data sources with minimal code.",
    category: "tech-stack"
  },
  {
    term: "ReAct Pattern",
    definition: "Reasoning and Acting pattern where agents alternate between thinking (reason) and doing (action).",
    category: "patterns"
  },
  {
    term: "Chain-of-Thought Prompting",
    definition: "Technique that encourages LLMs to break down complex problems into step-by-step reasoning.",
    category: "patterns"
  },
  {
    term: "Self-Reflection",
    definition: "Pattern where agents evaluate their own outputs, identify errors, and iteratively improve.",
    category: "patterns"
  },
  {
    term: "Multi-Agent System",
    definition: "Multiple specialized agents working together, each with defined roles, to solve complex problems.",
    category: "patterns"
  },
  {
```

```
  term: "Agent Orchestration",
  definition: "The system that connects models, memory, tools, and steps in order (e.g., LangChain, CrewAI, AutoGen).",
  category: "frameworks"
},
{
  term: "Memory (Short-term)",
  definition: "Current session or task history stored temporarily (e.g., episodic memory, working memory).",
  category: "fundamentals"
},
{
  term: "Memory (Long-term)",
  definition: "Persistent user preferences, past decisions, and knowledge stored across sessions (e.g., Vector DB, SQL, file sto
  category: "fundamentals"
},
{
  term: "Guardrails",
  definition: "Safety mechanisms that ensure agents behave responsibly, avoiding harmful or biased outputs.",
  category: "best-practices"
},
{
  term: "Multimodal AI",
  definition: "AI systems that can process multiple types of data: text, images, audio, video (e.g., GPT-4V, Gemini).",
  category: "advanced"
},
{
  term: "HyDE (Hypothetical Document Embeddings)",
  definition: "RAG technique where LLM generates hypothetical documents for better semantic search.",
  category: "rag"
},
{
  term: "Agentic Design Pattern",
  definition: "Architectural approaches for building agents: ReAct, Planning, Multi-Agent, Self-Reflection, etc.",
  category: "patterns"
},
{
  term: "System Prompt",
  definition: "Instructions that define agent's role, behavior, personality, and operational procedures.",
  category: "fundamentals"
},
{
  term: "Context Window",
  definition: "Maximum amount of text (in tokens) an LLM can process at once (e.g., 128k tokens for GPT-4).",
  category: "fundamentals"
},
```

```
  {
    term: "Token",
    definition: "Basic unit of text processing in LLMs, roughly equivalent to 4 characters or 0.75 words.",
    category: "fundamentals"
  },
  {
    term: "Fine-tuning",
    definition: "Training a pre-trained model on domain-specific data to improve performance for specialized tasks.",
    category: "advanced"
  },
  {
    term: "Prompt Engineering",
    definition: "Craft of designing effective prompts to guide AI behavior and improve output quality.",
    category: "best-practices"
  }
];
```

Design Patterns (6 main patterns)

javascript

```
const designPatterns = [
  {
    id: "react",
    name: "ReAct Agent",
    category: "Single-Agent",
    description: "A reasoning and acting framework where an agent alternates between reasoning (using LLMs) and acting (usi
    usedBy: "Most AI Agent Products",
    whenToUse: "General purpose tasks requiring flexible decision-making and tool use",
    diagram: "react-diagram.png",
    workflow: ["Query", "LLM (Reason)", "Tools (Action)", "Output"],
    pros: ["Simple to implement", "Flexible", "Works with most LLMs"],
    cons: ["Can be slow", "May loop unnecessarily"],
    codeExamples: {
      python: `from langchain.agents import create_react_agent
from langchain_openai import ChatOpenAI
from langchain.tools import WikipediaQueryRun

llm = ChatOpenAI(model="gpt-4o")
tools = [WikipediaQueryRun()]

agent = create_react_agent(
    llm=llm,
    tools=tools,
    prompt="You are a helpful research assistant"
)

result = agent.invoke({"input": "What is quantum computing?"})`,
      javascript: `import { ReActAgent } from "@langchain/agents";
import { ChatOpenAI } from "@langchain/openai";

const model = new ChatOpenAI({ model: "gpt-4o" });
const agent = new ReActAgent({ model, tools: [] });

const result = await agent.invoke({
  input: "What is quantum computing?"
});`,
      go: `// Go example using Go-LangChain
package main

import (
    "github.com/tmc/langchaingo/agents"
    "github.com/tmc/langchaingo/llms/openai"
)
```

```
func main() {
    llm := openai.New()
    agent := agents.NewReActAgent(llm, tools)
    result := agent.Run("What is quantum computing?")
}`,
      java: `// Java example using LangChain4j
import dev.langchain4j.agent.Agent;
import dev.langchain4j.model.openai.OpenAiChatModel;

OpenAiChatModel model = OpenAiChatModel.builder()
    .apiKey(System.getenv("OPENAI_API_KEY"))
    .modelName("gpt-4o")
    .build();

Agent agent = Agent.builder()
    .chatLanguageModel(model)
    .tools(tools)
    .build();

String result = agent.chat("What is quantum computing?");`
    },
    links: [
      { title: "ReAct Paper", url: "https://arxiv.org/abs/2210.03629" },
      { title: "LangChain ReAct Docs", url: "https://python.langchain.com/docs/modules/agents/agent_types/react" },
      { title: "Tutorial Video", url: "https://youtube.com/watch?v=..." }
    ]
  },
  {
    id: "codeact",
    name: "CodeAct Agent",
    category: "Single-Agent",
    description: "The CodeAct architecture by Manus AI allows agents to autonomously execute Python code instead of using
    usedBy: "Manus AI",
    whenToUse: "Complex data processing, mathematical computations, file operations",
    diagram: "codeact-diagram.png",
    workflow: ["User", "Agent (Think)", "CodeAct", "Environment", "Observation", "Result"],
    pros: ["Can handle complex logic", "More precise than text", "Native code execution"],
    cons: ["Security risks", "Requires sandboxing", "Limited LLM code quality"],
    codeExamples: {
      python: `# CodeAct pattern with code execution
from langchain_experimental.utilities import PythonREPL

python_repl = PythonREPL()
```

```
def code_agent(query):
    # Agent decides to write code
    code = llm.generate_code(query)

    # Execute code in sandbox
    result = python_repl.run(code)

    return result

# Example: "Calculate factorial of 10"
result = code_agent("Calculate factorial of 10")`,
      javascript: `// JavaScript with VM2 sandbox
const { VM } = require('vm2');

function codeAgent(query) {
  const code = llm.generateCode(query);

  const vm = new VM({ timeout: 1000 });
  const result = vm.run(code);

  return result;
}`
    },
    links: [
      { title: "CodeAct Paper", url: "https://arxiv.org/abs/2402.01030" },
      { title: "Manus AI", url: "https://manus.im" }
    ]
  },
  {
    id: "self-reflection",
    name: "Self-Reflection",
    category: "Self-Improving",
    description: "Self-Reflection or Reflexion involves self-evaluating outputs, using feedback loops to identify errors, and iter
    usedBy: "Open Serve AI",
    whenToUse: "High-quality outputs needed, writing tasks, complex problem solving",
    workflow: ["Query", "LLM", "Memory/Tools", "First draft", "Critique", "Generator", "Result"],
    codeExamples: {
      python: `from langchain.prompts import PromptTemplate

def self_reflection_agent(query):
    # Generate initial response
    draft = llm.invoke(query)
```

```
    # Self-critique
    critique_prompt = f"Critique this response: {draft}"
    critique = llm.invoke(critique_prompt)

    # Improve based on critique
    improve_prompt = f"Improve this based on critique:\\n{draft}\\n\\nCritique: {critique}"
    final = llm.invoke(improve_prompt)

    return final`
    }
  },
  {
    id: "multi-agent",
    name: "Multi-Agent Workflow",
    category: "Collaborative",
    description: "A Multi-Agent Workflow is a collaborative system where multiple specialized agents work together to build a
    usedBy: "Gemini Deep Research",
    whenToUse: "Complex tasks requiring specialized expertise, parallel processing",
    workflow: ["Query", "Agent 1 (Manager)", "Agent 2/3/n (Specialists)", "Aggregator LLM", "Output"],
    codeExamples: {
      python: `from crewai import Agent, Task, Crew

# Define specialized agents
researcher = Agent(
    role='Researcher',
    goal='Gather comprehensive information',
    tools=[search_tool]
)

analyst = Agent(
    role='Data Analyst',
    goal='Analyze and synthesize findings',
    tools=[analysis_tool]
)

writer = Agent(
    role='Writer',
    goal='Create final report',
    tools=[]
)

# Coordinate agents
crew = Crew(
    agents=[researcher, analyst, writer],
```

```
    tasks=[research_task, analysis_task, writing_task]
)

result = crew.kickoff()`
    }
  },
  {
    id: "agentic-rag",
    name: "Agentic RAG",
    category: "Advanced RAG",
    description: "Agentic RAG involves AI agents retrieving and evaluating relevant data to generate context-aware and well re
    usedBy: "Perplexity",
    whenToUse: "Question answering with citations, research tasks, fact-checking",
    workflow: ["Query", "Planning Agent", "Retrieval Agent", "Generator Agent", "Response Judge"],
    codeExamples: {
      python: `from langchain.agents import create_agent
from langchain.tools.retriever import create_retriever_tool

# Create retrieval tool
retriever_tool = create_retriever_tool(
    retriever=vector_store.as_retriever(),
    name="knowledge_base",
    description="Search the knowledge base"
)

# Planning agent decides queries
planner = create_agent(llm, [retriever_tool])

# Execute agentic RAG
result = planner.invoke({
    "input": "Compare quantum computing approaches"
})`
    }
  },
  {
    id: "planning",
    name: "Planning Pattern",
    category: "Goal-Oriented",
    description: "The AI breaks a high-level goal into smaller steps, executes them with the help of agents, and revises the plan
    whenToUse: "Multi-step projects, task decomposition, complex workflows",
    workflow: ["High-Level Goal", "Planning", "Generate Task", "Single Task Agent", "Adjusted Task", "Replan"],
    codeExamples: {
      python: `def planning_agent(goal):
    # Break down goal
```

```
    plan = llm.invoke(f"Break this into steps: {goal}")

    tasks = parse_plan(plan)
    results = []

    for task in tasks:
        result = execute_task(task)

        if not result.success:
            # Replan
            tasks = replan(task, result.error)
        else:
            results.append(result)

    return aggregate_results(results)`
    }
  }
];
```

Frameworks Data (Top 10)

javascript

```javascript
const frameworks = [
  {
    id: "langchain",
    name: "LangChain",
    logo: "langchain-logo.png",
    category: "Modular",
    tagline: "Build custom LLM agents using reusable components",
    description: "Modular framework to build complex, multi-turn conversational agents using LLMs and custom tools. Industr
    features: ["Agent executors", "Memory modules", "Tool integration", "Built-in templates"],
    bestFor: "Building complex, multi-turn conversational agents using LLMs and custom tools",
    pricing: "Free (Open Source)",
    languages: ["Python", "JavaScript"],
    stars: "90k+",
    difficulty: "Intermediate",
    officialLinks: {
      docs: "https://python.langchain.com",
      github: "https://github.com/langchain-ai/langchain",
      tutorials: "https://youtube.com/c/langchain"
    },
    codeExample: "// See design patterns section",
    pros: ["Extensive ecosystem", "Great documentation", "Large community"],
    cons: ["Can be complex", "Steep learning curve for beginners"],
    useCases: ["Chatbots", "RAG systems", "Document QA", "Agent workflows"]
  },
  // ... (add other 9 frameworks similarly)
];
```

Tech Stack Data (by layer)

```javascript
```

```
const techStack = {
  deployment: [
    { name: "Groq", category: "Infrastructure", description: "Ultra-fast LLM inference", url: "https://groq.com", pricing: "Pay-p
    { name: "AWS", category: "Cloud", description: "Amazon cloud services", url: "https://aws.amazon.com", pricing: "Pay-as
    { name: "Together.ai", category: "Inference", description: "Hosted model inference", url: "https://together.ai", pricing: "Pay
    // ... more tools
  ],
  evaluation: [
    { name: "LangSmith", category: "Monitoring", description: "LangChain observability", url: "https://smith.langchain.com",
    { name: "Weights & Biases", category: "MLOps", description: "Experiment tracking", url: "https://wandb.ai", pricing: "Fre
    // ... more tools
  ],
  // ... other layers
};
```

TECHNICAL IMPLEMENTATION REQUIREMENTS

HTML Structure

```
html
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Building Agentic Systems Guide</title>
  <style>
    /* Color Variables */
    :root {
      --primary-purple: #7C3AED;
      --accent-blue: #3B82F6;
      --teal: #06B6D4;
      --orange: #F97316;
      --dark-bg: #1E293B;
      --card-bg: #334155;
      --text-primary: #F1F5F9;
      --text-secondary: #94A3B8;
      --success: #10B981;
      --warning: #FBBF24;
    }

    * {
      margin: 0;
      padding: 0;
      box-sizing: border-box;
    }

    body {
      font-family: 'Inter', -apple-system, BlinkMacSystemFont, sans-serif;
      background: var(--dark-bg);
      color: var(--text-primary);
      line-height: 1.6;
    }

    /* Tab Navigation */
    .tabs {
      display: flex;
      gap: 1rem;
      padding: 1rem 2rem;
      background: linear-gradient(135deg, var(--primary-purple), var(--accent-blue));
      overflow-x: auto;
      position: sticky;
      top: 0;
```

```
    z-index: 100;
}

.tab {
    padding: 0.75rem 1.5rem;
    border: none;
    background: rgba(255, 255, 255, 0.1);
    color: white;
    cursor: pointer;
    border-radius: 8px;
    transition: all 0.3s;
    white-space: nowrap;
}

.tab:hover {
    background: rgba(255, 255, 255, 0.2);
    transform: translateY(-2px);
}

.tab.active {
    background: white;
    color: var(--primary-purple);
}

/* Main Content Area */
.content {
    max-width: 1400px;
    margin: 0 auto;
    padding: 2rem;
}

.tab-content {
    display: none;
}

.tab-content.active {
    display: block;
    animation: fadeIn 0.3s;
}

@keyframes fadeIn {
    from { opacity: 0; transform: translateY(10px); }
    to { opacity: 1; transform: translateY(0); }
}
```

```css
/* Card Styles */
.card {
    background: var(--card-bg);
    border-radius: 12px;
    padding: 2rem;
    margin-bottom: 2rem;
    border: 1px solid rgba(255, 255, 255, 0.1);
    transition: all 0.3s;
}

.card:hover {
    transform: translateY(-5px);
    box-shadow: 0 10px 30px rgba(124, 58, 237, 0.3);
}

/* Code Block Styles */
.code-block {
    background: #0f172a;
    border-radius: 8px;
    padding: 1rem;
    margin: 1rem 0;
    position: relative;
}

.language-tabs {
    display: flex;
    gap: 0.5rem;
    margin-bottom: 1rem;
}

.language-tab {
    padding: 0.5rem 1rem;
    background: rgba(255, 255, 255, 0.05);
    border: none;
    color: var(--text-secondary);
    cursor: pointer;
    border-radius: 4px;
}

.language-tab.active {
    background: var(--primary-purple);
    color: white;
}
```

```css
.copy-button {
    position: absolute;
    top: 1rem;
    right: 1rem;
    padding: 0.5rem 1rem;
    background: var(--accent-blue);
    border: none;
    color: white;
    border-radius: 4px;
    cursor: pointer;
}

/* Wizard Styles */
.wizard {
    max-width: 900px;
    margin: 0 auto;
}

.wizard-step {
    display: none;
}

.wizard-step.active {
    display: block;
}

.progress-bar {
    height: 4px;
    background: rgba(255, 255, 255, 0.1);
    border-radius: 2px;
    margin-bottom: 2rem;
}

.progress-fill {
    height: 100%;
    background: linear-gradient(90deg, var(--primary-purple), var(--teal));
    border-radius: 2px;
    transition: width 0.3s;
}

/* Search */
.search-box {
    width: 100%;
```

```
    padding: 1rem;
    background: var(--card-bg);
    border: 1px solid rgba(255, 255, 255, 0.1);
    border-radius: 8px;
    color: white;
    font-size: 1rem;
    margin-bottom: 2rem;
}

/* Modal for Graphics */
.modal {
    display: none;
    position: fixed;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    background: rgba(0, 0, 0, 0.9);
    z-index: 1000;
    justify-content: center;
    align-items: center;
}

.modal.active {
    display: flex;
}

.modal-content {
    max-width: 90%;
    max-height: 90%;
}

/* Comparison Matrix */
.comparison-table {
    width: 100%;
    border-collapse: collapse;
    margin: 2rem 0;
}

.comparison-table th,
.comparison-table td {
    padding: 1rem;
    border: 1px solid rgba(255, 255, 255, 0.1);
    text-align: left;
```

```
        }

        .comparison-table th {
            background: var(--primary-purple);
            position: sticky;
            top: 0;
        }

        /* Responsive */
        @media (max-width: 768px) {
            .tabs {
                padding: 1rem;
            }

            .content {
                padding: 1rem;
            }

            .card {
                padding: 1rem;
            }
        }
    </style>
</head>
<body>
    <!-- Navigation Tabs -->
    <div class="tabs">
        <button class="tab active" data-tab="home">Home</button>
        <button class="tab" data-tab="builder">⭐ Agent Builder</button>
        <button class="tab" data-tab="fundamentals">Fundamentals</button>
        <button class="tab" data-tab="patterns">Design Patterns</button>
        <button class="tab" data-tab="frameworks">Frameworks</button>
        <button class="tab" data-tab="advanced">Advanced Patterns</button>
        <button class="tab" data-tab="techstack">Tech Stack</button>
        <button class="tab" data-tab="best-practices">Best Practices</button>
        <button class="tab" data-tab="tools">Free Tools</button>
        <button class="tab" data-tab="graphics">Reference Graphics</button>
    </div>

    <!-- Main Content -->
    <div class="content">
        <!-- HOME TAB -->
        <div id="home" class="tab-content active">
            <!-- Home content here -->
```

```
    </div>

    <!-- AGENT BUILDER TAB -->
    <div id="builder" class="tab-content">
        <div class="wizard">
            <div class="progress-bar">
                <div class="progress-fill" style="width: 0%"></div>
            </div>

            <div class="wizard-step active" data-step="1">
                <!-- Step 1 content -->
            </div>

            <!-- More steps... -->
        </div>
    </div>

    <!-- Other tabs... -->
</div>

<!-- Graphics Modal -->
<div class="modal" id="graphicModal">
    <button class="close-btn">×</button>
    <button class="nav-btn prev">‹</button>
    <img class="modal-content" id="modalImage" src="" alt="">
    <button class="nav-btn next">›</button>
</div>

<script>
    // Tab switching
    const tabs = document.querySelectorAll('.tab');
    const tabContents = document.querySelectorAll('.tab-content');

    tabs.forEach(tab => {
        tab.addEventListener('click', () => {
            const targetId = tab.dataset.tab;

            tabs.forEach(t => t.classList.remove('active'));
            tab.classList.add('active');

            tabContents.forEach(content => {
                content.classList.remove('active');
                if (content.id === targetId) {
                    content.classList.add('active');
```

```
            }
        });

        // Save progress
        localStorage.setItem('lastTab', targetId);
    });
});

// Load graphics from CSV
async function loadGraphics() {
    // Read graphics.csv
    // Display as thumbnail grid
    // Add click handlers for modal
}

// Copy code functionality
function setupCopyButtons() {
    document.querySelectorAll('.copy-button').forEach(btn => {
        btn.addEventListener('click', async (e) => {
            const codeBlock = e.target.closest('.code-block');
            const code = codeBlock.querySelector('code').textContent;
            await navigator.clipboard.writeText(code);
            btn.textContent = 'Copied!';
            setTimeout(() => btn.textContent = 'Copy', 2000);
        });
    });
}

// Initialize app
document.addEventListener('DOMContentLoaded', () => {
    loadGraphics();
    setupCopyButtons();

    // Load last viewed tab
    const lastTab = localStorage.getItem('lastTab');
    if (lastTab) {
        document.querySelector(`[data-tab="${lastTab}"]`).click();
    }
});
</script>
</body>
</html>
```

READING GRAPHICS FROM CSV

The app must read `graphics.csv` which has this structure:

- FileName
- FilePath (relative path)
- Title

Use Papaparse to read the CSV:

```javascript
```

```
async function loadGraphicsFromCSV() {
    const response = await fetch('graphics.csv');
    const csvText = await response.text();

    Papa.parse(csvText, {
        header: true,
        skipEmptyLines: true,
        complete: (results) => {
            const graphics = results.data;
            renderGraphicsGallery(graphics);
        }
    });
}

function renderGraphicsGallery(graphics) {
    const gallery = document.getElementById('graphics-gallery');

    graphics.forEach((graphic, index) => {
        const thumbnail = document.createElement('div');
        thumbnail.className = 'graphic-thumbnail';
        thumbnail.innerHTML = `
            <img src="${graphic.FilePath}" alt="${graphic.Title}">
            <h4>${graphic.Title}</h4>
        `;

        thumbnail.addEventListener('click', () => {
            openGraphicModal(graphic, index, graphics);
        });

        gallery.appendChild(thumbnail);
    });
}

function openGraphicModal(graphic, index, allGraphics) {
    const modal = document.getElementById('graphicModal');
    const img = document.getElementById('modalImage');

    img.src = graphic.FilePath;
    img.alt = graphic.Title;
    modal.classList.add('active');

    // Navigation
    document.querySelector('.prev').onclick = () => {
```

```
    const newIndex = (index - 1 + allGraphics.length) % allGraphics.length;
    openGraphicModal(allGraphics[newIndex], newIndex, allGraphics);
  };

  document.querySelector('.next').onclick = () => {
    const newIndex = (index + 1) % allGraphics.length;
    openGraphicModal(allGraphics[newIndex], newIndex, allGraphics);
  };
}
```

PERSISTENCE WITH localStorage

Track user progress:

```javascript
```

```javascript
const AppState = {
    progress: {
        completedSections: [],
        bookmarks: [],
        journalEntries: [],
        lastTab: 'home'
    },

    save() {
        localStorage.setItem('agenticSystemsProgress', JSON.stringify(this.progress));
    },

    load() {
        const saved = localStorage.getItem('agenticSystemsProgress');
        if (saved) {
            this.progress = JSON.parse(saved);
        }
    },

    markComplete(sectionId) {
        if (!this.progress.completedSections.includes(sectionId)) {
            this.progress.completedSections.push(sectionId);
            this.save();
        }
    },

    addBookmark(item) {
        this.progress.bookmarks.push(item);
        this.save();
    },

    addJournalEntry(entry) {
        this.progress.journalEntries.push({
            date: new Date().toISOString(),
            ...entry
        });
        this.save();
    }
};
```

FINAL CHECKLIST

- ✅ All 10 tabs implemented
- ✅ Interactive Agent Builder Wizard with 14 steps
- ✅ Code examples in Python, JavaScript, TypeScript, Go, Java
- ✅ Language toggle for all code blocks
- ✅ Framework comparison matrix
- ✅ All 44 graphics displayed as clickable thumbnails
- ✅ Modal with prev/next navigation
- ✅ Read graphics from CSV file
- ✅ Progress tracking with localStorage
- ✅ Learning journal
- ✅ Search functionality
- ✅ Bookmarks/favorites
- ✅ Copy code buttons
- ✅ Resource links for all frameworks/tools
- ✅ Responsive design
- ✅ Dark mode with infographic color scheme

END OF PROMPT PART 2