COMPLETE PROMPT: Building Agentic Systems Guide - Interactive HTML Tutorial Application

APPLICATION OVERVIEW

Create a comprehensive, fully-functional HTML/CSS/JavaScript tutorial application titled **"Building Agentic Systems Guide"** that guides users through the complete process of building AI agents using RAG, multimodal processing, and various tools. This application will serve as an interactive learning platform with an agent builder wizard, comprehensive reference material, code examples in multiple languages, and clickable reference graphics.

DESIGN SPECIFICATIONS

Color Scheme

Extract colors from the uploaded infographics for visual continuity:

- **Primary Purple**: #7C3AED (from infographics)
- **Accent Blue**: #3B82F6
- **Teal/Cyan**: #06B6D4
- **Orange Accent**: #F97316
- **Dark Background**: #1E293B
- **Card Background**: #334155
- **Text Primary**: #F1F5F9
- **Text Secondary**: #94A3B8
- **Success Green**: #10B981
- **Warning Yellow**: #FBBF24

Visual Style

- Modern, gradient-heavy design matching infographic aesthetics
- Dark mode with vibrant accent colors
- Glassmorphism effects for cards
- Smooth transitions and hover effects

- Responsive grid layouts
- Sans-serif font (Inter or similar)

TAB STRUCTURE

1. Home/Welcome Tab

- Hero section with animated title
- Quick overview of what users will learn
- "Start Building" CTA button that opens Agent Builder tab
- Statistics: "10 Design Patterns | 4 Featured Frameworks | 20+ RAG Techniques"
- Visual journey map showing the learning path

2. Agent Builder Tab ⭐ (FEATURED)

Interactive step-by-step wizard that consolidates all step-by-step guides from the graphics into ONE comprehensive process:

The 10-Step Consolidated Process:

Step 1: Define Agent Purpose & Goals

- Text box: "Describe what your agent should do"
- Clarifying questions appear:
  - "Who are your target users?"
  - "What problem are you solving?"
  - "What deliverables should the agent produce?"
- Show example: "A sales assistant that qualifies leads, researches prospects, and drafts outreach emails"

Step 2: Select Input Sources

- Checkbox options:
  - ☐ Text data
  - ☐ Voice/Audio
  - ☐ Images/Video
  - ☐ API connections

        - ☐ Databases/CRMs
        - ☐ Real-time data streams
    - Dynamic UI shows selected options

Step 3: Data Preparation & Preprocessing

- Guide on cleaning and normalizing data
- Code snippets for:
    - Text chunking
    - Data formatting
    - Schema validation
- Tools: Pandas, Airbyte, Databricks

Step 4: Choose the Right LLM

- Interactive comparison table (from graphic 36):
    - GPT-4.1, Claude Sonnet 4, Gemini 2.5 Pro, Grok 4, Claude 3.5 Sonnet
    - Columns: Speed, Cost, Context Window, Best For
- Model selection dropdown
- Hosted vs Local options

Step 5: Design Agent Architecture

- Visual pattern selector:
    - ReAct Agent (reason → act → observe)
    - CodeAct Agent (execute code)
    - Multi-Agent Workflow
    - Self-Reflection
    - Agentic RAG
    - Planning Pattern
- Show architecture diagram for selected pattern

Step 6: Craft Prompts & System Instructions

- Prompt template editor

- Best practices sidebar
- "Test Prompt" button
- Examples from GPT-4o, Claude

Step 7: Add Memory & Context

- Memory type selector:
  - Short-term (episodic, working memory)
  - Long-term (vector DB, SQL, file store)
- Tools: Pinecone, Weaviate, ChromaDB, FAISS
- RAG integration toggle

Step 8: Enable Tool Use & External Actions

- Tool library with search:
  - APIs (web, apps, data)
  - Data tools (SQL, calculators, databases)
  - Action tools (email, CRM, ticket creation)
  - Orchestration tools (LangChain, OpenAI Function Calling)
- Custom tool creator

Step 9: Orchestrate Multi-Agent Collaboration (if needed)

- Single vs Multi-agent decision tree
- Orchestration options:
  - Sequential
  - Hierarchical
  - Manager-Agent pattern
  - Decentralized
- Framework selector: CrewAI, LangGraph, AutoGen, Swarm

Step 10: Add Multimodal Capabilities (Optional)

- Modality checklist:
  - Voice (Whisper, ElevenLabs, GPT-4o voice)

    - Vision (GPT-4V, Claude 3.5, Gemini)
    - Document processing (Document AI)
- Code examples for each

Step 11: Format & Deliver Outputs

- Output format selector:
    - JSON
    - Structured data
    - PDF/Reports
    - Human-readable text
    - Dashboards
- Tools: Pydantic, LangChain parsers, Plotly

Step 12: Testing & Validation

- Testing checklist:
    - Edge cases
    - User experience tests
    - Integration tests
    - Performance evaluation
- Tools: Postman, PyTest, Jupyter, MLflow

Step 13: Deploy with API or UI

- Deployment options:
    - Cloud (AWS, Azure, Vercel, Streamlit)
    - Docker/Kubernetes
    - FastAPI
    - Frontend (React, Gradio)
- Monitoring setup

Step 14: Monitor & Continuous Learning

- Monitoring metrics dashboard mockup

- Feedback loops
- Fine-tuning triggers
- Tools: MCP Logs, OpenAI Evals, Weights & Biases, Grafana

After wizard completion:

- Generate a downloadable PDF "Agent Blueprint"
- Show recommended frameworks based on selections
- Provide starter code templates
- Link to relevant documentation

---

3. Fundamentals Tab

Section: What is an AI Agent?

- Definition from graphic 5
- The 4 Pillars diagram:
    1. LLM (The Brain) - with examples
    2. Perception (Senses) - text, images, audio, video
    3. Tools (Hands) - APIs, calculators, functions
    4. Memory (Notebook) - short-term + long-term
- Traits: Reason, Plan, Act, Learn, Adapt, Delegate
- Evolution diagram: Simple Prompting → RAG → AI Agents

Section: Types of AI Agents (Graphic 23) 5 types with detailed workflows:

1. **Self-Directed AI Agents** - Define goal → Perceive → Plan → Execute → Observe → Self-correct
2. **Collaborative Multi-Agent Systems** - Assign roles → Share context → Divide sub-tasks → Exchange results → Merge outcomes → Optimized output
3. **Cognitive AI Agents** - Perceive → Retrieve memory → Reason → Generate → Evaluate → Store learnings
4. **Tool-Augmented AI Agents** - Receive task → Identify required tools → Connect via API → Fetch/process → Validate → Return enriched response

5. **Reflective (Self-Improving) Agents** - Execute → Analyze outcome → Identify improvements → Adjust reasoning → Update memory → Improve accuracy

Section: 7 Pillars of Agentic AI (Graphic 24) Visual pillar diagram with details:

1. **Autonomy** - Tools: AutoGen, CrewAI, LangGraph, OpenAgents, MetaGPT, AgentVerse
2. **Goal-Directed Planning** - Tools: AutoGPT, BabyAGI, ReAct, LangChain Agent Executors, Camel, DUST
3. **Communication & Collaboration** - Tools: AutoGen, CrewAI, LangGraph, ChatDev, SupaAgent, AgentHub
4. **Reasoning & Decision Making** - Tools: GPT-4o, Claude 3 Opus, Mistral, Chain-of-Thought, Self-Ask, OpenDevin, Thought Source
5. **Tool Use & Environment Interaction** - Tools: LangChain Toolkits, Function Calling, BrowserPilot, WebAgent, ToolLLM, CrewAI Tools
6. **Memory & Learning** - Tools: LangChain Memory, MemGPT, LlamaIndex, Pinecone, Chroma, Qdrant, Weaviate, MemoryGraph
7. **Safety, Alignment & Evaluation** - Tools: Guardrails AI, Constitutional AI, OpenAI Moderation API, Red-Teaming Agents, TruLens, Helicone

Section: Agent Skills (Graphic 26) On-demand expertise wheel: UI, Database, Testing, API, Tintips, Backend, Deploy

---

4. Design Patterns Tab

Interactive cards for 6 main patterns (Graphics 6, 20, 22):

Pattern 1: ReAct Agent

- Visual diagram: Query → LLM (Reason) ↔ Tools (Action) → Output
- Description: "A reasoning and acting framework where an agent alternates between reasoning (using LLMs) and acting (using tools like Google or email)"
- Used by: Most AI Agent Products
- When to use: General purpose, need flexible decision-making
- Code example toggle (Python/JavaScript/TypeScript/Go)

Pattern 2: CodeAct Agent

- Visual diagram: User → Agent (Think) → CodeAct → Environment → Observation → Result
- Description: "The CodeAct architecture by Manus AI allows agents to autonomously execute Python code instead of using JSON, enabling them to handle complex tasks more efficiently"
- Used by: Manus
- When to use: Complex data processing, mathematical computation tasks
- Code example

Pattern 3: Modern Tool Use

- Visual diagram: Query → Agent → MCP Server 1/2 (APIs)
- Description: "Enables agents to leverage tools like Kagi Search, AWS and others using MCP for enhanced functionality with very little code execution"
- Used by: Cursor
- Code example with MCP integration

Pattern 4: Self-Reflection

- Visual diagram: Query → LLM → Memory/Tools → First draft → Critique → Generator → Result
- Description: "Self-Reflection or Reflexion involves self-evaluating outputs, using feedback loops to identify errors, and iteratively improving via learning or critiques, enabling adaptation"
- Used by: Open Serve AI
- Code example

Pattern 5: Multi-Agent Workflow

- Visual diagram: Query → Agent 1 ↔ Agent 2/3/n ↔ Aggregator LLM → Output
- Description: "A Multi-Agent Workflow is a collaborative system where multiple specialized agents work together to build a comprehensive output with greater precision than single architectures"
- Used by: Gemini Deep Research
- Code example with agent coordination

Pattern 6: Agentic RAG

- Visual diagram: Query → Planning Agent → Retrieval Agent → Generator Agent → Response Judge
- Description: "Agentic RAG involves AI agents retrieving and evaluating relevant data to generate context-aware and well reasoned output using memory and tools"

- Used by: Perplexity
- Code example with vector database

Additional Pattern: Planning Pattern

- High-level goal → Planning → Generate Task → Execute → Single Task Agent → Adjusted Task → Replan → Completed Tasks

---

5. Frameworks Tab (Top 10 with Top 4 featured)

Featured Framework Cards (LangChain, CrewAI, AutoGen, LangGraph):

LangChain

- **Logo/Icon**
- **Category**: Modular Framework
- **Description**: "Modular framework to build custom LLM agents using reusable components. Industry standard with 90k+ GitHub stars."
- **Key Features**:
  - 🔧 Agent executors
  - 🧠 Memory modules
  - 📎 Tool integration
  - 📊 Built-in templates
- **Best For**: Building complex, multi-turn conversational agents using LLMs and custom tools
- **Pricing**: Free (Open Source)
- **Languages**: Python, JavaScript
- **Official Links**:
  - 🌐 Official Docs
  - 💻 GitHub
  - 📺 Tutorial
- **Code Example** (toggleable languages):

```
python
```

```python
from langchain.agents import create_agent
from langchain_community.tools import WikipediaQueryRun
from langchain_openai import ChatOpenAI

# Initialize model and tools
model = ChatOpenAI(model="gpt-4o")
tools = [WikipediaQueryRun()]

# Create agent
agent = create_agent(
    model=model,
    tools=tools,
    system_prompt="You are a helpful research assistant"
)

# Run agent
result = await agent.invoke({
    "messages": [{"role": "user", "content": "Research quantum computing"}]
})
```

javascript

```javascript
// JavaScript/TypeScript
import { createAgent } from "langchain";
import { ChatOpenAI } from "@langchain/openai";

const model = new ChatOpenAI({ model: "gpt-4o" });
const agent = createAgent({
  model,
  tools: [],
  systemPrompt: "You are a helpful assistant"
});

await agent.invoke({
  messages: [{ role: "user", content: "Hello!" }]
});
```

CrewAI

- **Category**: Collaborative Multi-Agent
- **Description**: "Multi-agent system with role assignment and task coordination. Ideal for building agent teams with clear role-based task delegation."

- **Key Features**:
  - 👥 Role distribution
  - 🎯 Agent teamwork
  - 📋 Task dependencies
  - 🔄 Sequential/hierarchical processes
- **Best For**: Teams and creative enterprises, managing agents with specialized roles
- **Pricing**: Free OSS + Enterprise plans
- **Languages**: Python
- **Official Links**:
  - 🌐 Docs
  - 💻 GitHub
  - 🎓 Learn
- **Code Example**:

```python
```

```python
from crewai import Agent, Task, Crew

# Define agents
researcher = Agent(
    role='Senior Researcher',
    goal='Uncover groundbreaking technologies',
    backstory='Expert in emerging tech trends',
    verbose=True
)

writer = Agent(
    role='Tech Writer',
    goal='Create engaging content about tech',
    backstory='Skilled technical writer',
    verbose=True
)

# Define tasks
research_task = Task(
    description='Research latest AI agents',
    agent=researcher,
    expected_output='Detailed research report'
)

write_task = Task(
    description='Write article about findings',
    agent=writer,
    expected_output='Engaging article'
)

# Create crew
crew = Crew(
    agents=[researcher, writer],
    tasks=[research_task, write_task],
    verbose=True
)

result = crew.kickoff()
```

AutoGen (Microsoft)

- **Category**: Event-Driven Multi-Agent

- **Description**: "Microsoft's framework for event-driven, distributed multi-agent systems. v0.4 uses actor model for scalability."
- **Key Features**:
  - 🔄 Asynchronous messaging
  - 🌐 Cross-language support (Python/.NET)
  - 🧩 Modular architecture
  - 🔌 Extensible with plugins
- **Best For**: Experimental framework creating fully autonomous agents that plan and execute tasks without human input
- **Pricing**: Free (Open Source)
- **Languages**: Python, .NET
- **Official Links**:
  - 🌐 Docs
  - 💻 GitHub
  - 🎬 Research
- **Code Example**:

```python
from autogen_agentchat.agents import AssistantAgent
from autogen_ext.models.openai import OpenAIChatCompletionClient

# Create model client
model_client = OpenAIChatCompletionClient(model="gpt-4.1")

# Create agent
agent = AssistantAgent(
    "assistant",
    model_client=model_client,
    system_message="You are a helpful AI assistant"
)

# Run agent
await agent.run(task="Help me plan my day")
```

LangGraph

- **Category**: Reactive Graph-Based
- **Description**: "Graph-based execution model for reactive stateful flows. Low-level orchestration for complex workflows."
- **Key Features**:
  - 📊 Node-based task flow
  - 🔄 Cycles and retries
  - 💾 Built-in persistence
  - 🧑‍💻 Human-in-the-loop
- **Best For**: Excellent for memory and long-heavy logic, complex multi-step workflows with conditional branching
- **Pricing**: Free (OSS) + LangSmith Platform
- **Languages**: Python, JavaScript
- **Official Links**:
  - 🌐 Docs
  - 💻 GitHub
  - 🎓 Academy
- **Code Example**:

```python
```

```python
from langgraph.graph import START, StateGraph
from typing_extensions import TypedDict

class State(TypedDict):
    text: str

def node_a(state: State) -> dict:
    return {"text": state["text"] + "a"}

def node_b(state: State) -> dict:
    return {"text": state["text"] + "b"}

# Build graph
graph = StateGraph(State)
graph.add_node("node_a", node_a)
graph.add_node("node_b", node_b)
graph.add_edge(START, "node_a")
graph.add_edge("node_a", "node_b")

# Execute
result = graph.compile().invoke({"text": ""})
# {'text': 'ab'}
```

Other Framework Cards (condensed):

- **n8n** - Low-code workflow automation
- **Make** - Scenarios, sub-scenarios
- **LangGraph** (duplicate - remove)
- **AgentOps** - Monitoring
- **Superagent** - Open-source sandbox
- **Haystack Agents** - DEV-centric, RAG pipelines

Framework Comparison Matrix: Interactive table with sortable columns:

Framework	No-Code	LLM Support	MCP	Tools	Agent Orchestration	Best Use Case
LangChain	❌	Remote	✅	Predefined	Tool-calling only	Complex workflows
CrewAI	❌	Remote	✅	Predefined	Threads	Agent teams
AutoGen	❌	Remote	❌	Predefined	Flow-based	Multi-agent collab
LangGraph	❌	Local, Remote	✅	LangChain Functions	Graph (DAG)	Reactive workflows
n8n	✅	Local, Remote	✅	100+ integrations	Workflows	Multi-turn LLM
Make	✅	Limited	⚠️	100+ integrations	Scenarios	Task automation

---

6. Advanced Patterns Tab (RAG + Multimodal)

Section: RAG vs Agentic RAG (Graphics 25, 34) Side-by-side comparison:

Traditional RAG:

- User Prompt → Similarity Search → Indexed Docs → Augmented Prompt → LLM → Response
- Limitations: No multi-step, retrieval happens once

Agentic RAG:

- User Prompt → Query Planning Agent → Retrieval Agent (rewrites user query) → Retrieved & Reranked Results → Generator Agent → Response Judge
- Loop: Rewritten with Feedback
- Supports: Multi-turn reasoning, query rewriting, tool use, feedback loops

Section: 20 Different Types of RAG Techniques (Graphic 35) Grid of cards with descriptions:

1. Standard RAG
2. Corrective RAG
3. Speculative RAG

4. Fusion RAG
5. Agentic RAG
6. Self RAG
7. Interactive RAG
8. Contextual RAG
9. Multi-Source RAG
10. Hierarchical RAG
11. Multi-Pass RAG
12. Feedback RAG
13. Adversarial RAG
14. Auto RAG
15. Hybrid RAG
16. REALM
17. RAPTOR
18. CRAG
19. RePLUG
20. ATLAS

Each card includes:

- Diagram thumbnail
- 2-sentence description
- "When to use" tag

Section: HyDE (Hypothetical Document Embeddings) (Graphic 39)

- Comparison diagram: Traditional RAG vs HyDE
- HyDE process:
    1. Query → LLM → Hypothetical text
    2. Encode hypothetical text
    3. Similarity search with hypothetical text
    4. Retrieve context + Query + Hypothetical → Prompt → LLM → Response

Section: Multimodal Agents (Graphic 16) The Data Pipeline:

- Old Approach: Chunking → Captions → Analyzer → OCR text
- ❌ Issues: Object Detect, Summarize Images, Don't align

New Approach (Unified VLM):

- Multimodality (Unified VLM): Screenshot Page → Unified Vector → Input (PDF/Slide Deck)
- ✅ Pass to VLM → Unified Vector

Tech Stack:

- Language: Python
- LLM: Gemini 2.0 Flash (Multimodal)
- Embedding Model: Voyage AI Multimodal 3 (VLM)
- Vector Store: ChromaDB, Pinecone
- Orchestration: LangChain
- Database: Memory + Vector Store

The Agent Execution Workflow:

1. User Query
2. Agent Logic (LLM): Query Engine + LLM summarize
3. Tools (web search, calculator, APIs)
4. Tool Execution → Response
5. Synthesis (Multimodal LLM)
6. Update Memory
7. Session Manager

Key Takeaways:

- Simple Prompting vs RAG vs AI Agents (2025)
- Summary: Screenshot Page, Pass to VLM, Unified Vector

---

7. Tech Stack Tab

Organize by 8 layers from Agentic AI Tech Stack (Graphic 3):

Layer 1: Deployment and Infrastructure

- groq, AWS, together.ai, baseten, Modal, Fireworks AI, Replicate, Google Cloud

Layer 2: Evaluation and Monitoring

- LangSmith, mlflow, Weights & Biases, deepchecks, Fairlearn, Holistic AI, HuggingFace Evals

Layer 3: Foundation Models

- Claude 3.7 Sonnet, Mistral AI, cohere, Claude 4, Gemini 2.5 Pro, LLAMA 4, GPT-4o

Layer 4: Orchestration Frameworks

- LangChain, DSPy, Microsoft AutoGen, adalflow, LlamaIndex, Haystack, LiteLLM, Dify, Semantic Kernel, RAY

Layer 5: Vector Databases

- Milvus, qdrant, redis, pgvector, Vald, chroma, Pinecone, elasticsearch, Vespa, Weaviate

Layer 6: Embedding Models

- Voyage AI, NOMIC, fastText, Hugging Face, OpenAI, spaCy, VECTOR FLOW, cohere

Layer 7: Data Ingestion and Extraction

- Scrapy, docling, Beautifulsoup, DIFFBOT, Firecrawl, LLAMAPARSE, Amazon Textract, Apache Tika

Layer 8: Memory and Context Management

- Letta, mem0, zep, chroma, cognee, LangChain, LlamaIndex

Each tool should have:

- Logo/icon
- Brief description
- Official link
- Pricing tier indicator (Free/Paid/Enterprise)

---

8. Best Practices Tab

Section: 8 Things to Keep in Mind (Graphic 30) Interactive checklist:

1. ✅ Agentic Design Patterns (Reflection, Tool Use, Planning, MultiAgent)
2. ✅ Framework Choice (LangGraph vs LangChain vs CrewAI)
3. ✅ Tool Use from Scratch (User → ToolAgent (ModeTool) → Hacker News API → JSON Response)
4. ✅ Start Small, Handle sensitive data, Secure First
5. ✅ Avoid Hallucinations via RAG
6. ✅ Prompt Engineering (Good prompts lead to better performance)
7. ✅ Fallback Strategy (Handle errors gracefully)
8. ✅ Performance Budget (Balance cost, speed, task accuracy)

Section: 10 Mistakes to Avoid (Graphic 37) Visual "mistakes" wheel with outcomes:

1. ❌ Skipping Clear Goal Definition → Agent handles random tasks
2. ❌ Poor Prompt Experience & Inconsistent Answers → Clunky, confusing, underused agent
3. ❌ No Integration with External Tools → Agent can't act, limited usefulness
4. ❌ Not Using a Vector Database → Agent doesn't recall, outdated, hallucinated response
5. ❌ Not Using a Database → Agent starts fresh, irrelevant data
6. ❌ Overloading with Too Many Tools → Too much Autonomy
7. ❌ Foregoing Modular Design → Difficult to scale, rigid output
8. ❌ Ignoring Evaluation Metrics → Can't measure, don't know performance
9. ❌ No Feedback Loop or Learning Mechanism → Agent never improves
10. ❌ Overloading Guardrails & Safety → Harmful, biased, unsafe outputs

Section: Model Selection Guide (Graphic 36) "Which LLM Should Power Your AI Agents?"

- Comparison table with results (80%+ success rate)
- Available tools icons
- Model comparison: GPT-4.1, Claude 4 mini, Command mini, Grok 4, Gemini 2.5 Flash, Kimi K2, Claude Sonnet 4

- Best practices: Use non-reasoning LLMs for planning, provide clear tools
- Insights: 4:1 × 4:1-mini, Claude beats 83% cheaper, Above 2 tools autonomy breaks

Section: Agent Development Lifecycle (Graphics 10, 14) 6 Stages:

1. **Planning**: Draft Business Needs → Define Agent Objectives → Resource Allocation → Risk + Ethics Review
2. **Design**: Design Guardrails → Grounding With Context → Select a model → Choose a Framework
3. **Development**: Build the agent logic → Integrate the models → Fine-tune Models (if Required) → Document Setup
4. **Testing**: Test Edge Cases → Conduct User Experience Tests → Perform Integration Tests → Evaluate Performance
5. **Deployment**: Launch Agent → Make sure Guardrails are working → Observability → Compliance Validation
6. **Maintenance**: Act upon User feedback → Optimize Operations → Monitor Agent Objectives

---

9. Free Tools Tab (Graphic 29)

Organize by category:

Chatbot I: ChatGPT, Bard, Bing **Chatbot II**: Forefront, Poe, Perplexity **Design**: Adobe Firefly, Bing, Microsoft Designer **Video**: Capcut, Descript, Visla.us **Content**: Opus Clip, Cohesive, Copy.ai **Presentation**: Tome, Decktopus, Gamma **Productivity**: Saga.so, Taskade, Kickresume **Code**: Codeium, AskCodi, CodeWhisperer **Meetings**: Loopin, Krisp, tl;dv

Each tool card:

- Logo
- Category tag
- Brief description
- Link

---

10. Reference Graphics Tab

Interactive Gallery:

- Thumbnail grid of all 44 graphics (4 columns on desktop, 2 on tablet, 1 on mobile)
- Click thumbnail → Opens full-size modal
- Modal features:
  - Full-size image
  - Title (from graphics.csv)
  - "Download" button
  - Navigation arrows (prev/next)
  - Close button
- Search/filter by keyword

Graphics organization: Read from the CSV file (`graphics.csv`) which contains:

- FileName
- FilePath (relative path)
- Title

Display thumbnails with titles. Clicking opens modal with full image.

Make sure image paths are relative (same folder as HTML file).

INTERACTIVE FEATURES

All Features Implementation:

1. Interactive Agent Builder Wizard ✅

- 14-step guided experience
- Text input for agent description
- Dynamic questions based on inputs
- Real-time recommendations
- Progress indicator
- Generate downloadable PDF blueprint

2. Code Examples with Language Toggle ✅

- Every code example has language tabs: Python | JavaScript | TypeScript | Go | Java
- Copy button for each code block
- Syntax highlighting
- Line numbers

3. Framework Comparison Matrix ✅

- Sortable columns
- Filter by features
- Highlight differences
- Export to CSV

4. Progress Tracking ✅

- Track reviewed sections
- Mark code examples as "tried"
- Overall completion percentage
- Persist in localStorage

5. Learning Journal ✅

- Date picker
- Time spent slider
- Confidence level (1-5 stars)
- Notes textarea
- View past entries
- Export journal

6. Search Functionality ✅

- Global search across all tabs
- Highlight matches
- Jump to results
- Search history

7. Difficulty Filtering ✅

- Beginner/Intermediate/Advanced tags
- Filter content by level
- Recommended learning path

8. Bookmarks/Favorites ✅

- Bookmark any section, framework, pattern
- Favorites sidebar
- Quick access
- Sync across sessions

9. Copy Code Button ✅

- One-click copy
- Success feedback
- Works on all code blocks

10. Resource Links ✅

- Every framework/tool has:
  - 🌐 Official Docs
  - 💻 GitHub Repo
  - 📺 Tutorial Videos
  - 📖 Related Articles

DATA STRUCTURE

Definitions (20-30 items)

```javascript
```

```
const definitions = [
  {
    term: "AI Agent",
    definition: "An intelligent system that can independently plan, make decisions, execute tasks, and adapt to changes—minim
    category: "fundamentals"
  },
  {
    term: "LLM (Large Language Model)",
    definition: "The brain of an agent - a powerful model capable of reasoning, planning, and generating step-by-step outputs (e
    category: "fundamentals"
  },
  {
    term: "RAG (Retrieval-Augmented Generation)",
    definition: "Enhances LLMs by retrieving relevant documents from an external knowledge base and injecting them into the
    category: "rag"
  },
  {
    term: "Agentic RAG",
    definition: "Extends RAG using LLM-powered agents that can plan
```