Page 1: Introduction to Node.js

Node.js is a runtime environment built on Chrome's V8 Engine.

It enables developers to execute JavaScript on the server side.

Its event-driven, non-blocking architecture makes it highly scalable

and well-suited for applications that involve heavy I/O operations,

such as web servers, realtime applications, and LLM pipelines.

Page 2: Event Loop Deep Dive

The event loop is the core of Node.js. It manages asynchronous operations without blocking the main thread. Timers, I/O callbacks, microtasks, and event queues work together to execute code efficiently. Understanding the event loop helps developers write optimized LangChain pipelines that handle documents, embeddings, and vector operations asynchronously.

Page 3: Node.js Modules and ES Imports

Node.js offers two module systems: CommonJS (require) and ES Modules (import).
Most LangChain.js modern examples use ES Modules. Libraries like langchain,
@langchain/community, vector DB clients, and LLM SDKs are imported using
the ES syntax, making the code cleaner and compliant with modern standards.

Page 4: Working with Files and Buffers

PDFs, text files, and other document formats are essential in RAG pipelines. Node.js provides fs, streams, and buffer utilities to handle large files efficiently. These capabilities are especially useful when chunking documents, parsing PDFs, or indexing knowledge bases for LangChain applications.

Page 5: PDF Processing in Node.js

Several libraries such as pdf-parse, pdfjs, and LangChain's PDFLoader allow extracting text from PDF files. LangChain.js simplifies document-loading processes. Example:

```
import { PDFLoader } from "langchain/document_loaders/fs/pdf";
const loader = new PDFLoader("file.pdf");
const docs = await loader.load();
```

This output can be chunked and embedded.

Page 6: Introduction to LangChain.js

LangChain.js is the JavaScript version of the LangChain framework.

It provides modules for LLMs, document loaders, embeddings, vector stores,

chains, memory, and agents. Developers use LangChain.js to create search engines,

AI assistants, chatbots, and knowledge-based applications using Node.js.

Page 7: Building a RAG Pipeline with Node.js

A standard RAG pipeline includes:

1. Load documents (PDF, TXT, HTML)
2. Split into chunks
3. Generate embeddings
4. Store in vector DB (FAISS, Pinecone, Qdrant)
5. Query using an LLM

Node.js performs these operations asynchronously, improving performance and scalability for enterprise-level AI systems.

Page 8: Embeddings and Vector Databases

Embeddings are numerical representations of text.

LangChain.js supports OpenAI, HuggingFace, Cohere, and more.

Vector DB options in Node.js include:

- Pinecone
- Qdrant
- FAISS (Python-only)
- Chroma

These databases enable semantic search across large document sets.

Page 9: Agents in LangChain.js

Agents allow LLMs to make decisions, call tools, and use reasoning loops.

Node.js supports tools such as:

- Web search
- Code execution
- Database queries
- Custom APIs

LangGraph.js enhances this by providing stateful, multi-step agent workflows

with memory and decision trees.

Page 10: LangGraph Overview

LangGraph allows building agentic workflows with nodes, edges, and loops.

Unlike simple chains, LangGraph can:

- Maintain state
- Retry failed steps
- Follow conditional logic
- Execute iterative reasoning

This is ideal for research agents, chatbot loops, or automated data labeling systems built in Node.js.

Page 11: Integrating OpenAI or Gemini

LangChain.js integrates directly with LLM APIs. Example:

```
import { ChatOpenAI } from "@langchain/openai";
const llm = new ChatOpenAI({ apiKey: process.env.OPENAI_API_KEY });
```

These models help generate responses, summarize documents, evaluate search results, and build conversational agents in Node.js applications.

Page 12: Summary & Best Practices

Node.js is an excellent choice for RAG and agentic workflows because:

- Non-blocking architecture

- Large ecosystem of AI libraries

- Excellent performance for indexing

- Works seamlessly with LangChain.js & LangGraph

Best practices:

- Use async/await for I/O tasks

- Stream large PDFs instead of loading fully

- Use environment variables for keys

- Keep embeddings batch-optimized

This knowledge prepares you to build advanced AI systems in Node.js.