NeuroLearn Architecture Documentation

Table of Contents

System Overview

NeuroLearn is built on a modern, scalable architecture designed for intelligent educational content delivery:

```
┌─────────────────────────────────────────────────────────────────┐
│                         CLIENT LAYER                            │
│  ┌──────────┐  ┌──────────┐  ┌──────────┐  ┌──────────┐        │
│  │   Chat   │  │  Teach   │  │  Notes   │  │Flashcards│        │
│  │   Page   │  │   Page   │  │   Page   │  │   Page   │        │
│  └────┬─────┘  └────┬─────┘  └────┬─────┘  └────┬─────┘        │
└───────┼─────────────┼─────────────┼─────────────┼──────────────┘
        │             │             │             │
        ▼             ▼             ▼             ▼
┌─────────────────────────────────────────────────────────────────┐
│                       API LAYER (Next.js)                       │
│  ┌──────────┐  ┌──────────┐  ┌──────────┐  ┌──────────┐        │
│  │/api/chat │  │/api/teach│  │/api/notes│  │/api/flash│        │
│  └────┬─────┘  └────┬─────┘  └────┬─────┘  └────┬─────┘        │
└───────┼─────────────┼─────────────┼─────────────┼──────────────┘
        │             │             │             │
        ▼             ▼             ▼             ▼
┌─────────────────────────────────────────────────────────────────┐
│                      SERVICE LAYER                              │
│  ┌───────────────────────────────────────────────────────────┐  │
│  │                  MULTI-AGENT ORCHESTRATOR                 │  │
│  │  ┌─────────┐  ┌─────────┐  ┌─────────┐  ┌─────────┐      │  │
│  │  │ Content │  │  Image  │  │  Video  │  │  Quiz   │      │  │
│  │  │  Agent  │  │  Agent  │  │  Agent  │  │  Agent  │      │  │
│  │  └─────────┘  └─────────┘  └─────────┘  └─────────┘      │  │
│  └───────────────────────────────────────────────────────────┘  │
│  ┌───────────────────────────────────────────────────────────┐  │
│  │                   PERSONA SYSTEM                          │  │
│  │    Einstein | Newton | Curie | Turing | Darwin | ...      │  │
│  └───────────────────────────────────────────────────────────┘  │
└─────────────────────────────────────────────────────────────────┘
        │             │             │             │
        ▼             ▼             ▼             ▼
┌─────────────────────────────────────────────────────────────────┐
│                     EXTERNAL SERVICES                           │
│  ┌──────────┐  ┌──────────┐  ┌──────────┐  ┌──────────┐        │
│  │  OpenAI  │  │  DALL-E  │  │  Tavily  │  │  Gemini  │        │
│  │  GPT-4o  │  │    3     │  │  Search  │  │   API    │        │
│  └──────────┘  └──────────┘  └──────────┘  └──────────┘        │
└─────────────────────────────────────────────────────────────────┘
        │
        ▼
┌─────────────────────────────────────────────────────────────────┐
│                      DATA LAYER                                 │
│  ┌───────────────────────────────────────────────────────────┐  │
│  │              PostgreSQL + Prisma ORM                      │  │
│  │  Sessions | Messages | Notes | Concepts | Flashcards      │  │
│  └───────────────────────────────────────────────────────────┘  │
└─────────────────────────────────────────────────────────────────┘
```

Multi-Agent Orchestration

Overview

The Multi-Agent system is the core intelligence layer, enabling parallel processing of complex educational content generation.

Agent Types

```typescript
// lib/teaching-agents.ts

interface Agent {
  name: string;
  role: string;
  capabilities: string[];
  execute: (context: Context) => Promise<Result>;
}

const AGENTS = {
  CONTENT_AGENT: {
    name: 'Content Generator',
    role: 'Create structured educational content',
    capabilities: ['text_generation', 'explanation', 'examples']
  },
  IMAGE_AGENT: {
    name: 'Visual Content Creator',
    role: 'Generate educational diagrams and illustrations',
    capabilities: ['dalle_generation', 'web_search', 'fallback']
  },
  VIDEO_AGENT: {
    name: 'Video Curator',
    role: 'Find relevant YouTube educational videos',
    capabilities: ['youtube_search', 'relevance_scoring']
  },
  QUIZ_AGENT: {
    name: 'Assessment Generator',
    role: 'Create adaptive quiz questions',
    capabilities: ['question_generation', 'difficulty_scaling']
  },
  MINDMAP_AGENT: {
    name: 'Knowledge Mapper',
    role: 'Create concept relationship diagrams',
    capabilities: ['mermaid_generation', 'concept_linking']
  },
  REALWORLD_AGENT: {
    name: 'Current Events Connector',
    role: 'Link concepts to real-world applications',
    capabilities: ['news_search', 'application_examples']
  }
};
```

Orchestration Flow

```
async function generateAdaptiveLesson(topic: string, persona: Persona) {
  // Phase 1: Context Building
  const contextPack = await buildContextPack(topic, persona);

  // Phase 2: Parallel Agent Execution
  const [content, images, videos, realWorld] = await Promise.all([
    contentAgent.generate(contextPack),
    imageAgent.generate(contextPack),      // 3-tier fallback
    videoAgent.search(contextPack),        // Tavily YouTube search
    realWorldAgent.connect(contextPack)    // Current events
  ]);

  // Phase 3: Quiz Generation (depends on content)
  const quiz = await quizAgent.generate(content, contextPack);

  // Phase 4: Mind Map Generation
  const mindMap = await mindMapAgent.generate(content);

  // Phase 5: Assembly
  return assembleLesson({
    content,
    images,
    videos,
    realWorld,
    quiz,
    mindMap,
    persona
  });
}
```

Performance Metrics

Metric	Before Optimization	After Optimization
Total Generation Time	90+ seconds	~54 seconds
Sequential Calls	6	1 (parallel)
API Timeout	30s (failures)	300s (stable)

MCP (Model Context Protocol)

Context Pack Architecture

MCP enables consistent context sharing across all agents:

```
// lib/context-packs.ts

interface ContextPack {
  // Student Profile
  student: {
    id: string;
    learningStyle: 'visual' | 'auditory' | 'kinesthetic';
    currentLevel: 'beginner' | 'intermediate' | 'advanced';
    preferences: string[];
  };

  // Persona Context
  persona: {
    id: string;
    name: string;
    teachingStyle: string;
    expertise: string[];
    systemPrompt: string;
  };

  // Topic Context
  topic: {
    subject: string;
    mainConcept: string;
    prerequisites: string[];
    relatedTopics: string[];
  };

  // Session Context
  session: {
    previousTopics: string[];
    masteredConcepts: string[];
    strugglingAreas: string[];
  };
}
```

Context Flow

```
┌──────────────────────────────────────────────────────────────┐
│                     CONTEXT BUILDER                           │
│                                                               │
│  User Query ──► Topic Analysis ──► Persona Selection          │
│        │              │                    │                  │
│        ▼              ▼                    ▼                  │
│  ┌─────────┐   ┌──────────┐        ┌────────────┐            │
│  │ Session │   │ Subject  │        │  Teaching  │            │
│  │ History │   │ Keywords │        │   Style    │            │
│  └────┬────┘   └────┬─────┘        └─────┬──────┘            │
│       │             │                    │                    │
│       └─────────────┼────────────────────┘                    │
│                     ▼                                         │
│             ┌──────────────┐                                  │
│             │ CONTEXT PACK │                                  │
│             └──────┬───────┘                                  │
│                    │                                          │
│    ┌───────────────┼────────────────┐                         │
│    ▼               ▼                ▼                         │
│ ┌──────┐       ┌──────┐        ┌──────┐                       │
│ │Agent1│       │Agent2│        │Agent3│                       │
│ └──────┘       └──────┘        └──────┘                       │
│                                                              │
└──────────────────────────────────────────────────────────────┘
```

Persona System Prompts

```
// lib/personas.ts

export function getPersonaSystemPrompt(persona: Persona): string {
  return `You are ${persona.name}, ${persona.title}.

Your Expertise: ${persona.expertise.join(', ')}

Teaching Philosophy: ${persona.teachingStyle}

Personality: ${persona.personality}

When responding:
1. Stay in character as ${persona.name}
2. Use examples from your field of expertise
3. Reference your famous works and discoveries
4. Maintain your unique communication style
5. Be encouraging and adapt to the student's level

Remember: "${persona.quote}"`;
}
```

Frontend Architecture

Component Hierarchy

```
app/
├── layout.tsx              # Root layout with navigation
├── page.tsx                # Landing page with persona showcase
├── chat/
│   └── page.tsx            # Persona-aware chat interface
├── teach/
│   └── page.tsx            # Lesson generation UI
├── notes/
│   └── page.tsx            # Second Brain interface
├── flashcards/
│   └── page.tsx            # SM-2 review system
└── dashboard/
    └── page.tsx            # Analytics view
```

Key Components

Message Bubble (`components/message-bubble.tsx`)

```
interface MessageBubbleProps {
  content: string;
  role: 'user' | 'assistant';
  agentType?: AgentType;
  index: number;
}

export function MessageBubble({ content, role, agentType, index }:
MessageBubbleProps) {
  return (
    <motion.div
      initial={{ opacity: 0, y: 20 }}
      animate={{ opacity: 1, y: 0 }}
      transition={{ delay: index * 0.1 }}
      className={cn(
        'rounded-2xl p-4 max-w-[80%]',
        role === 'user'
          ? 'bg-purple-600 ml-auto'
          : 'bg-gray-800 mr-auto'
      )}
    >
      {agentType && <AgentAvatar type={agentType} />}
      <ReactMarkdown>{content}</ReactMarkdown>
    </motion.div>
  );
}
```

Persona Selector (`app/chat/page.tsx`)

```
{PERSONAS.map((persona) => (
  <button
    key={persona.id}
    onClick={() => setSelectedPersona(persona)}
    className={cn(
      'rounded-xl border transition-all',
      selectedPersona?.id === persona.id
        ? 'border-purple-500 bg-purple-900/30'
        : 'border-gray-700 hover:border-gray-600'
    )}
  >
    <Image
      src={persona.image}
      alt={persona.name}
      width={48}
      height={48}
      className="rounded-full"
    />
    <span>{persona.name}</span>
  </button>
))}
```

State Management

```
// Chat state with streaming support
const [messages, setMessages] = useState<Message[]>([]);
const [isLoading, setIsLoading] = useState(false);
const [currentAgent, setCurrentAgent] = useState<AgentType | null>(null);
const [selectedPersona, setSelectedPersona] = useState<Persona | null>(null);

// Streaming response handling
const reader = response.body?.getReader();
while (true) {
  const { done, value } = await reader?.read();
  if (done) break;

  const chunk = decoder.decode(value);
  // Parse SSE format and update messages
}
```

Backend Architecture

API Route Structure

```typescript
// app/api/chat/route.ts - Streaming Chat API

export async function POST(req: Request) {
  const { messages, personaId, modelType } = await req.json();

  // Build persona-aware system prompt
  const persona = getPersona(personaId);
  const systemPrompt = getPersonaSystemPrompt(persona);

  // Route to appropriate agent
  const agent = routeQuestion(messages[messages.length - 1].content);

  // Stream response
  const stream = new ReadableStream({
    async start(controller) {
      const encoder = new TextEncoder();

      // Send agent info
      controller.enqueue(encoder.encode(
        `data: ${JSON.stringify({ type: 'start', agentType: agent })}\n\n`
      ));

      // Stream OpenAI response
      const response = await openai.chat.completions.create({
        model: 'gpt-4o',
        messages: [{ role: 'system', content: systemPrompt }, ...messages],
        stream: true
      });

      for await (const chunk of response) {
        const content = chunk.choices[0]?.delta?.content;
        if (content) {
          controller.enqueue(encoder.encode(
            `data: ${JSON.stringify({ type: 'content', content })}\n\n`
          ));
        }
      }

      controller.close();
    }
  });

  return new Response(stream, {
    headers: { 'Content-Type': 'text/event-stream' }
  });
}
```

Lesson Generation API

```
// app/api/teach/generate/route.ts

export const maxDuration = 300; // 5-minute timeout for complex generation

export async function POST(req: Request) {
  const { topic, studentId, personaId } = await req.json();

  // Build context pack
  const contextPack = await buildContextPack(topic, studentId, personaId);

  // Execute multi-agent orchestration
  const lesson = await generateAdaptiveLesson(contextPack);

  return NextResponse.json(lesson);
}
```

3-Tier Image Generation

Implementation

```
// lib/image-generator.ts

export async function generateEducationalImage(prompt: string): Promise<ImageResult> {
  // Tier 1: DALL-E 3 (Primary)
  const dalleResult = await generateWithDALLE(prompt);
  if (dalleResult) {
    return { url: dalleResult, source: 'dalle', prompt };
  }

  // Tier 2: Tavily Web Search (Fallback)
  const tavilyResult = await searchWithTavily(prompt);
  if (tavilyResult) {
    return { url: tavilyResult, source: 'tavily', prompt };
  }

  // Tier 3: Gemini (Last Resort)
  const geminiResult = await generateWithGemini(prompt);
  if (geminiResult) {
    return { url: geminiResult, source: 'gemini', prompt };
  }

  // Tier 4: Placeholder
  return {
    url: `https://placehold.co/800x600?text=${encodeURIComponent(prompt)}`,
    source: 'fallback',
    prompt
  };
}
```

Fallback Logic

```
┌──────────────────────────────────────────────────────────────┐
│                    IMAGE GENERATION REQUEST                   │
└──────────────────────────────────────────────────────────────┘
                          │
                          ▼
                  ┌───────────────┐
                  │   DALL-E 3    │
                  │   (Primary)   │
                  └───────┬───────┘
                          │
              ┌───────────┴───────────┐
              │ Success?              │
              └───────────┬───────────┘
                    │           │
                   YES          NO
                    │           │
                    ▼           ▼
              ┌─────────┐ ┌───────────────┐
              │ Return  │ │    Tavily     │
              │  Image  │ │  Web Search   │
              └─────────┘ └───────┬───────┘
                                  │
                      ┌───────────┴───────────┐
                      │ Success?              │
                      └───────────┬───────────┘
                            │           │
                           YES          NO
                            │           │
                            ▼           ▼
                      ┌─────────┐ ┌───────────────┐
                      │ Return  │ │    Gemini     │
                      │  Image  │ │  (Fallback)   │
                      └─────────┘ └───────┬───────┘
                                          │
                              ┌───────────┴───────────┐
                              │ Success?              │
                              └───────────┬───────────┘
                                    │           │
                                   YES          NO
                                    │           │
                                    ▼           ▼
                              ┌─────────┐ ┌───────────────┐
                              │ Return  │ │  Placeholder  │
                              │  Image  │ │    (Final)    │
                              └─────────┘ └───────────────┘
```

Database Design

Entity Relationship Diagram

```
┌─────────────────┐     ┌─────────────────┐
│   ChatSession   │     │     Student     │
├─────────────────┤     ├─────────────────┤
│ id              │     │ id              │
│ createdAt       │     │ name            │
│ updatedAt       │     │ learningStyle   │
│ personaId       │     │ currentLevel    │
└────────┬────────┘     └─────────────────┘
         │
         │ 1:N
         ▼
┌─────────────────┐     ┌─────────────────┐
│     Message     │     │      Note       │
├─────────────────┤     ├─────────────────┤
│ id              │     │ id              │
│ role            │     │ title           │
│ content         │     │ content         │
│ agentType       │     │ subject         │
│ sessionId (FK)  │     │ tags (JSON)     │
└─────────────────┘     └────────┬────────┘
                                 │
                   ┌─────────────┼─────────────┐
                   │             │             │
                   ▼             ▼             ▼
          ┌──────────────┐ ┌──────────────┐ ┌──────────────┐
          │   Concept    │ │  Flashcard   │ │     Quiz     │
          ├──────────────┤ ├──────────────┤ ├──────────────┤
          │ id           │ │ id           │ │ id           │
          │ name         │ │ front        │ │ question     │
          │ description  │ │ back         │ │ options      │
          │ noteId (FK)  │ │ difficulty   │ │ answer       │
          └──────┬───────┘ │ noteId (FK)  │ │ noteId (FK)  │
                 │         └──────┬───────┘ └──────────────┘
                 │                │
                 │                ▼
                 │         ┌─────────────────┐
                 │         │ FlashcardReview │
                 │         ├─────────────────┤
                 │         │ id              │
                 │         │ quality (0-5)   │
                 │         │ easeFactor      │
                 │         │ interval        │
                 │         │ nextReviewAt    │
                 │         │ flashcardId(FK) │
                 │         └─────────────────┘
                 │
                 ▼
    ┌───────────────────────┐
    │ ConceptRelationship   │
    ├───────────────────────┤
    │ id                    │
    │ fromConceptId (FK)    │
    │ toConceptId (FK)      │
    │ relationshipType      │
    └───────────────────────┘
```

Performance Optimizations

1. Parallel Execution

```
// Before: Sequential (90s+)
const content = await generateContent();
const images = await generateImages();
const videos = await searchVideos();

// After: Parallel (~54s)
const [content, images, videos] = await Promise.all([
  generateContent(),
  generateImages(),
  searchVideos()
]);
```

2. API Timeout Configuration

```
export const maxDuration = 300; // 5-minute timeout
```

3. Image Caching Strategy

```
// Response includes source tracking
return { imageUrl: result.url, source: result.source };
// Enables client-side caching decisions
```

4. Streaming Responses

```
// Server-Sent Events for real-time updates
const stream = new ReadableStream({ /* ... */ });
return new Response(stream, {
  headers: { 'Content-Type': 'text/event-stream' }
});
```

References

- Next.js App Router (https://nextjs.org/docs/app)
- OpenAI API Documentation (https://platform.openai.com/docs/api-reference)
- Prisma ORM (https://www.prisma.io/docs)
- Tavily Search API (https://docs.tavily.com/)
- SM-2 Spaced Repetition Algorithm (https://www.supermemo.com/en/archives1990-2015/english/ol/sm2)
- Framer Motion (https://www.framer.com/motion/)
- Tailwind CSS (https://tailwindcss.com/docs)

Last Updated: January 2026
Version: 2.0.0