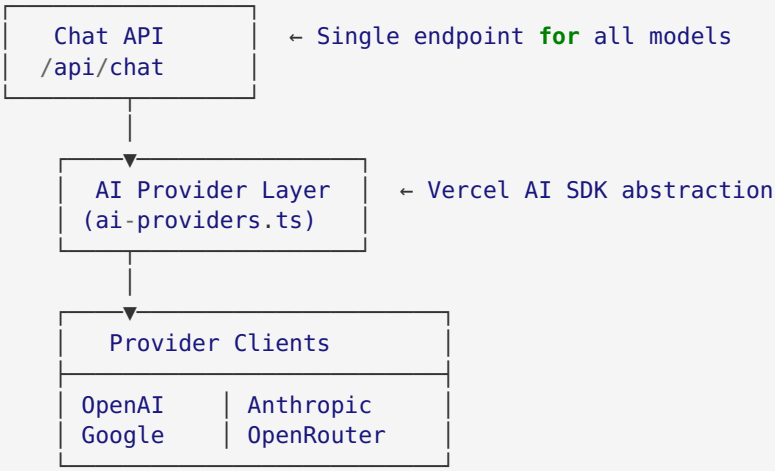
AI Provider System Documentation

Overview

This project uses the **Vercel AI SDK** to provide a unified interface for multiple AI providers. This enables seamless switching between models from OpenAI, Anthropic, Google, and others through a consistent API.

Architecture



Supported Providers

1. OpenAI

Models: GPT-4o, O1, O1-mini, GPT-4o-mini

Best For: Advanced reasoning, coding, multimodal tasks

Setup:

```
# Get API key from https://platform.openai.com/api-keys
OPENAI_API_KEY="sk-..."
```

Features:
- Extended context (128K-200K tokens)
- Vision capabilities
- Function calling
- Structured outputs
- Reasoning models (O1/O3)

2. Anthropic

Models: Claude 3.5 Sonnet, Claude 3 Opus, Claude 3 Haiku

Best For: Instruction-following, creative writing, complex analysis

Setup:

```
# Get API key from https://console.anthropic.com/settings/keys
ANTHROPIC_API_KEY="sk-ant-..."
```

Features:
- Massive context (200K tokens)
- Excellent instruction-following
- Safe and helpful outputs
- Vision capabilities

3. Google AI

Models: Gemini 2.0 Flash, Gemini 1.5 Pro, Gemini 1.5 Flash

Best For: Massive context (2M tokens), multimodal, speed

Setup:

```
# Get API key from https://aistudio.google.com/app/apikey
GOOGLE_GENERATIVE_AI_API_KEY="..."
```

Features:
- Huge context window (up to 2M tokens)
- Native multimodal support
- Fast inference
- Function calling
- Currently free in preview

4. OpenRouter (Recommended)

Models: 100+ models from all providers

Best For: Access to multiple providers with single API key

Setup:

```
# Get API key from https://openrouter.ai/keys
OPENROUTER_API_KEY="sk-or-..."
```

Features:
- Single API key for all models
- Automatic fallbacks
- Cost optimization
- Access to: DeepSeek V3, Llama 3.3, Qwen, and more

Usage

Basic Chat Completion

```
import { streamChat } from '@/lib/ai-providers'

const stream = await streamChat({
  model: 'gpt-4o',
  messages: [
    { role: 'user', content: 'Hello!' }
  ],
  temperature: 0.7,
  maxTokens: 1000,
})

// Stream to response
for await (const chunk of stream) {
  console.log(chunk)
}
```

With Tool Calling

```
import { generateChat } from '@/lib/ai-providers'
import { z } from 'zod'

const result = await generateChat({
  model: 'gpt-4o',
  messages: [
    { role: 'user', content: 'What is the weather in Tokyo?' }
  ],
  tools: [
    {
      name: 'getWeather',
      description: 'Get current weather for a location',
      parameters: z.object({
        location: z.string(),
        unit: z.enum(['celsius', 'fahrenheit']),
      }),
      execute: async (args) => {
        // Fetch weather data
        return { temp: 22, condition: 'sunny' }
      },
    },
  ],
})
```

Structured Output

```
import { generateStructured } from '@/lib/ai-providers'
import { z } from 'zod'

const schema = z.object({
  name: z.string(),
  age: z.number(),
  interests: z.array(z.string()),
})

const result = await generateStructured({
  model: 'gpt-4o',
  messages: [
    { role: 'user', content: 'Tell me about Marie Curie' }
  ],
  schema,
})

// result is typed and validated!
console.log(result.name, result.age, result.interests)
```

API Endpoints

POST /api/chat

Main chat endpoint - Handles streaming chat completions

Request:

```
{
  "projectId": "string",
  "message": "string",
  "model": "gpt-4o",
  "temperature": 0.7,
  "maxTokens": 4096
}
```

Response: Streaming text (Server-Sent Events)

GET /api/models

List available models - Get information about all models

Query Parameters:
- `provider`: Filter by provider (openai, anthropic, google, openrouter)
- `capability`: Filter by capability (code, vision, reasoning, etc.)
- `category`: Filter by category (flagship, fast, specialized)

Response:

```
{
  "models": [...],
  "categories": {...},
  "providers": {...},
  "availableProviders": [...],
  "statistics": {...}
}
```

Model Selection Guide

For Coding Tasks

1. **GPT-4o** - Best overall for code generation and debugging
2. **Claude 3.5 Sonnet** - Excellent for refactoring and code review
3. **DeepSeek V3** - Specialized coding model (via OpenRouter)
4. **O1-mini** - Advanced reasoning for complex algorithms

For Reasoning Tasks

1. **O1** - Advanced reasoning with “thinking” steps
2. **Claude 3 Opus** - Deep analysis and research
3. **GPT-4o** - General reasoning and problem-solving

For Speed/Cost

1. **Gemini 2.0 Flash** - Free, fast, capable (currently in preview)
2. **GPT-4o-mini** - Fast and affordable
3. **Gemini 1.5 Flash** - Great balance
4. **Claude 3 Haiku** - Fastest from Anthropic

For Long Documents

1. **Gemini 1.5 Pro** - 2M token context
2. **Claude 3.5 Sonnet** - 200K token context
3. **GPT-4o** - 128K token context

For Vision Tasks

1. **GPT-4o** - Best multimodal understanding
2. **Gemini 2.0 Flash** - Fast and capable
3. **Claude 3.5 Sonnet** - Good vision + reasoning

Configuration

Environment Variables

All API keys are optional - configure only the providers you want to use:

```
# .env
OPENAI_API_KEY=""              # For OpenAI models
ANTHROPIC_API_KEY=""           # For Anthropic models
GOOGLE_GENERATIVE_AI_API_KEY="" # For Google models
OPENROUTER_API_KEY=""          # For OpenRouter (recommended)
```

Adding a New Model

1. Add model configuration to `lib/ai-models.ts`:

```
{
  id: 'new-model',
  name: 'New Model',
  provider: 'openai',
  modelId: 'gpt-4o-2024-05-13',
  description: 'Description...',
  contextWindow: 128000,
  maxOutput: 4096,
  capabilities: ['text', 'code', 'vision'],
  bestFor: ['coding', 'reasoning'],
  pricing: {
    input: 2.5,
    output: 10.0
  }
}
```

1. The model is automatically available through the API!

Adding a New Provider

1. Install the Vercel AI SDK provider package:

```
npm install @ai-sdk/provider-name
```

1. Add provider client in `lib/ai-providers.ts`:

```
static getNewProvider() {
  const apiKey = process.env.NEW_PROVIDER_API_KEY
  if (!apiKey) throw new Error('API key not found')
  return createNewProvider({ apiKey })
}
```

1. Update `getLanguageModel()` to handle the new provider
2. Add provider info to `lib/ai-models.ts`

Best Practices

1. Model Selection

- Use **GPT-4o** or **Claude 3.5 Sonnet** for most tasks
- Use **O1** for complex reasoning (but it's slower and more expensive)
- Use **mini** models for high-volume, simple tasks
- Use **Gemini 2.0 Flash** for speed (free in preview!)

2. Cost Optimization

- Start with cheaper models and upgrade only when needed
- Use OpenRouter for automatic provider selection
- Cache system prompts when possible
- Set appropriate `maxTokens` limits

3. Error Handling

- Always check API key availability before making requests
- Implement fallback providers for reliability
- Handle rate limits gracefully
- Log errors for debugging

4. Security

- Never expose API keys in client-side code
- Use environment variables for all keys
- Implement rate limiting on your endpoints
- Validate and sanitize user inputs

Troubleshooting

“API key not found” Error

Solution: Configure the required API key in `.env` file

Model Not Streaming

Solution: Ensure you’re using `streamChat()` instead of `generateChat()`

Rate Limit Errors

Solution:
- Wait and retry with exponential backoff
- Switch to a different provider
- Use OpenRouter for automatic fallback

High Costs

Solution:
- Use mini models for simpler tasks
- Reduce `maxTokens` parameter
- Implement request caching
- Monitor usage with provider dashboards

Resources

- Vercel AI SDK Docs (https://sdk.vercel.ai/docs)
- OpenAI API Reference (https://platform.openai.com/docs)
- Anthropic API Reference (https://docs.anthropic.com)
- Google AI Studio (https://ai.google.dev)
- OpenRouter Documentation (https://openrouter.ai/docs)

Examples

See additional research files for comprehensive code examples and implementation patterns.