职场说话保命神器 - 实施计划

> **For Claude:** REQUIRED SUB-SKILL: Use superpowers:executing-plans to implement this plan task-by-task.

Goal: 构建一个React Web应用，将用户的"气话"转换为5种不同语气的职场表达版本

Architecture: 单页Web应用，使用React + Vite，Context API管理状态，localStorage持久化数据，直接调用第三方AI模型API

Tech Stack: React 18, Vite 5, localStorage, Fetch API, Lucide React Icons

前置准备

Task 0: 初始化项目

Files:

- Create: `package.json`
- Create: `vite.config.js`
- Create: `index.html`
- Create: `.gitignore`
- Create: `src/main.jsx`
- Create: `src/App.jsx`
- Create: `src/App.css`

Step 1: 初始化Git仓库

```
git init
```

Step 2: 创建package.json

创建文件 `package.json`：

```
{
  "name": "workplace-speech-saver",
  "private": true,
  "version": "1.0.0",
  "type": "module",
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "preview": "vite preview"
  },
  "dependencies": {
    "react": "^18.3.1",
    "react-dom": "^18.3.1"
  },
  "devDependencies": {
    "@vitejs/plugin-react": "^4.3.4",
    "vite": "^6.0.11"
  }
}
```

Step 3: 创建vite.config.js

创建文件 `vite.config.js`：

```
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'

export default defineConfig({
  plugins: [react()],
  server: {
    port: 3000
  }
})
```

Step 4: 创建index.html

创建文件 `index.html`：

```html
<!DOCTYPE html>
<html lang="zh-CN">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>职场说话保命神器</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>
```

Step 5: 创建.gitignore

创建文件 `.gitignore`：

```
# Logs
logs
*.log
npm-debug.log*

# Dependencies
node_modules/

# Build outputs
dist/
dist-ssr/

# Local env files
.env
.env.local
```

Step 6: 创建src/main.jsx

创建文件 `src/main.jsx`：

```
import { StrictMode } from 'react'
import { createRoot } from 'react-dom/client'
import './App.css'
import App from './App.jsx'

createRoot(document.getElementById('root')).render(
  <StrictMode>
    <App />
  </StrictMode>,
)
```

Step 7: 创建基础App组件

创建文件 `src/App.jsx`：

```
function App() {
  return (
    <div className="app">
      <h1>职场说话保命神器</h1>
    </div>
  )
}

export default App
```

Step 8: 创建基础样式

创建文件 `src/App.css`：

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

body {
  font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto', sans-serif;
  background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
  min-height: 100vh;
}

.app {
  max-width: 1200px;
  margin: 0 auto;
  padding: 20px;
}

h1 {
  text-align: center;
  color: white;
  margin-bottom: 40px;
  font-size: 2.5rem;
}
```

Step 9: 安装依赖并测试

```
npm install
npm run dev
```

Expected: Vite服务器启动在 http://localhost:3000，显示"职场说话保命神器"标题

Step 10: 提交初始代码

```
git add .
git commit -m "feat: initialize project with Vite + React"
```

核心功能实现

Task 1: 创建Context和状态管理

Files:

- Create: `src/contexts/AppContext.jsx`

Step 1: 创建Context Provider

创建文件 `src/contexts/AppContext.jsx`：

```jsx
import { createContext, useContext, useState, useEffect } from 'react'

const AppContext = createContext()

export const useApp = () => {
  const context = useContext(AppContext)
  if (!context) {
    throw new Error('useApp must be used within AppProvider')
  }
  return context
}

export const AppProvider = ({ children }) => {
  // API Keys
  const [apiKeys, setApiKeys] = useState(() => {
    const saved = localStorage.getItem('apiKeys')
    return saved ? JSON.parse(saved) : {
      openai: '',
      qianwen: '',
      wenxin: '',
      deepseek: ''
    }
  })

  // Selected Model
  const [selectedModel, setSelectedModel] = useState(() => {
    return localStorage.getItem('selectedModel') || 'openai'
  })

  // History
  const [history, setHistory] = useState(() => {
    const saved = localStorage.getItem('history')
    return saved ? JSON.parse(saved) : []
  })

  // Conversion Result
  const [conversionResult, setConversionResult] = useState(null)

  // Loading State
  const [loading, setLoading] = useState(false)

  // Error State
  const [error, setError] = useState(null)
```

```
// Save API Keys to localStorage
useEffect(() => {
  localStorage.setItem('apiKeys', JSON.stringify(apiKeys))
}, [apiKeys])

// Save selected model to localStorage
useEffect(() => {
  localStorage.setItem('selectedModel', selectedModel)
}, [selectedModel])

// Save history to localStorage
useEffect(() => {
  localStorage.setItem('history', JSON.stringify(history))
}, [history])

// Update API Key
const updateApiKey = (model, key) => {
  setApiKeys(prev => ({
    ...prev,
    [model]: key
  }))
}

// Add to history
const addToHistory = (input, results, model) => {
  const newItem = {
    id: Date.now().toString(),
    input,
    results,
    model,
    timestamp: Date.now()
  }
  setHistory(prev => [newItem, ...prev].slice(0, 50)) // Keep last 50
}

// Clear history
const clearHistory = () => {
  setHistory([])
  localStorage.removeItem('history')
}

const value = {
```

```
    apiKeys,
    selectedModel,
    setSelectedModel,
    history,
    conversionResult,
    setConversionResult,
    loading,
    setLoading,
    error,
    setError,
    updateApiKey,
    addToHistory,
    clearHistory
  }

  return (
    <AppContext.Provider value={value}>
      {children}
    </AppContext.Provider>
  )
}
```

Step 2: 更新App.jsx使用Context

修改文件 `src/App.jsx`：

```
import { AppProvider } from './contexts/AppContext'

function App() {
  return (
    <AppProvider>
      <div className="app">
        <h1>职场说话保命神器</h1>
      </div>
    </AppProvider>
  )
}

export default App
```

Step 3: 提交Context实现

```
git add src/contexts/AppContext.jsx src/App.jsx
git commit -m "feat: implement AppContext with state management"
```

Task 2: 创建输入组件

Files:

- Create: `src/components/InputSection.jsx`
- Create: `src/components/InputSection.css`

Step 1: 创建输入组件样式

创建文件 `src/components/InputSection.css`：

```css
.input-section {
  background: white;
  border-radius: 12px;
  padding: 24px;
  box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
  margin-bottom: 24px;
}

.input-section h2 {
  font-size: 1.25rem;
  color: #333;
  margin-bottom: 16px;
}

.model-selector {
  margin-bottom: 16px;
}

.model-selector label {
  display: block;
  font-weight: 600;
  color: #555;
  margin-bottom: 8px;
}

.model-selector select {
  width: 100%;
  padding: 10px 12px;
  border: 2px solid #e0e0e0;
  border-radius: 8px;
  font-size: 1rem;
  cursor: pointer;
  transition: border-color 0.2s;
}

.model-selector select:focus {
  outline: none;
  border-color: #667eea;
}

.textarea-wrapper {
  position: relative;
  margin-bottom: 16px;
```

```css
}

.textarea-wrapper textarea {
  width: 100%;
  min-height: 120px;
  padding: 12px;
  border: 2px solid #e0e0e0;
  border-radius: 8px;
  font-size: 1rem;
  font-family: inherit;
  resize: vertical;
  transition: border-color 0.2s;
}

.textarea-wrapper textarea:focus {
  outline: none;
  border-color: #667eea;
}

.char-count {
  position: absolute;
  bottom: 8px;
  right: 12px;
  font-size: 0.875rem;
  color: #999;
}

.button-group {
  display: flex;
  gap: 12px;
}

.convert-button {
  flex: 1;
  padding: 12px 24px;
  background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
  color: white;
  border: none;
  border-radius: 8px;
  font-size: 1rem;
  font-weight: 600;
  cursor: pointer;
  transition: transform 0.1s, box-shadow 0.2s;
```

```css
}

.convert-button:hover:not(:disabled) {
  transform: translateY(-2px);
  box-shadow: 0 4px 12px rgba(102, 126, 234, 0.4);
}

.convert-button:active:not(:disabled) {
  transform: translateY(0);
}

.convert-button:disabled {
  opacity: 0.6;
  cursor: not-allowed;
}

.convert-button.loading {
  display: flex;
  align-items: center;
  justify-content: center;
  gap: 8px;
}

.spinner {
  width: 16px;
  height: 16px;
  border: 2px solid rgba(255, 255, 255, 0.3);
  border-top-color: white;
  border-radius: 50%;
  animation: spin 0.6s linear infinite;
}

@keyframes spin {
  to { transform: rotate(360deg); }
}

.examples {
  margin-top: 16px;
  padding-top: 16px;
  border-top: 1px solid #e0e0e0;
}

.examples h3 {
```

```
  font-size: 0.875rem;
  color: #666;
  margin-bottom: 8px;
}

.example-buttons {
  display: flex;
  flex-wrap: wrap;
  gap: 8px;
}

.example-button {
  padding: 6px 12px;
  background: #f5f5f5;
  border: 1px solid #e0e0e0;
  border-radius: 6px;
  font-size: 0.875rem;
  cursor: pointer;
  transition: background 0.2s;
}

.example-button:hover {
  background: #e8e8e8;
}
```

Step 2: 创建输入组件

创建文件 `src/components/InputSection.jsx`：

```jsx
import { useState } from 'react'
import { useApp } from '../contexts/AppContext'
import './InputSection.css'

const EXAMPLES = [
  { text: '这个方案根本行不通！', label: '方案质疑' },
  { text: '你们怎么总是这样？', label: '团队抱怨' },
  { text: '我不干了！', label: '情绪爆发' },
  { text: '这谁写的代码？', label: '代码质疑' },
  { text: '又是你的错！', label: '指责他人' }
]

function InputSection() {
  const { selectedModel, setSelectedModel, loading } = useApp()
  const [input, setInput] = useState('')

  const handleSubmit = () => {
    if (input.trim()) {
      // TODO: Trigger conversion
      console.log('Converting:', input)
    }
  }

  const handleKeyDown = (e) => {
    if (e.key === 'Enter' && (e.ctrlKey || e.metaKey)) {
      handleSubmit()
    }
  }

  return (
    <div className="input-section">
      <h2>输入你的气话</h2>

      <div className="model-selector">
        <label>选择AI模型</label>
        <select
          value={selectedModel}
          onChange={(e) => setSelectedModel(e.target.value)}
        >
          <option value="openai">OpenAI (GPT-4)</option>
          <option value="qianwen">通义千问</option>
          <option value="wenxin">文心一言</option>
          <option value="deepseek">Deepseek</option>
```

```
  </select>
</div>

<div className="textarea-wrapper">
  <textarea
    value={input}
    onChange={(e) => setInput(e.target.value.slice(0, 500))}
    onKeyDown={handleKeyDown}
    placeholder="在这里输入你想说的气话...（Ctrl+Enter 快速提交）"
    maxLength={500}
  />
  <span className="char-count">{input.length}/500</span>
</div>

<div className="button-group">
  <button
    className="convert-button"
    onClick={handleSubmit}
    disabled={!input.trim() || loading}
  >
    {loading ? (
      <>
        <span className="spinner"></span>
        转换中...
      </>
    ) : (
      '转换'
    )}
  </button>
</div>

<div className="examples">
  <h3>试试这些示例：</h3>
  <div className="example-buttons">
    {EXAMPLES.map((example, index) => (
      <button
        key={index}
        className="example-button"
        onClick={() => setInput(example.text)}
      >
        {example.label}
      </button>
    ))}
```

```
        </div>
      </div>
    </div>
  )
}

export default InputSection
```

Step 3: 在App.jsx中使用InputSection

修改文件 `src/App.jsx`：

```
import { AppProvider } from './contexts/AppContext'
import InputSection from './components/InputSection'
import './App.css'

function App() {
  return (
    <AppProvider>
      <div className="app">
        <h1>职场说话保命神器</h1>
        <InputSection />
      </div>
    </AppProvider>
  )
}

export default App
```

Step 4: 测试输入组件

```
npm run dev
```

Expected: 在浏览器中看到输入框、模型选择器和示例按钮

Step 5: 提交输入组件

```
git add src/components/InputSection.jsx src/components/InputSection.css src/App.jsx
git commit -m "feat: implement InputSection component"
```

Task 3: 创建API调用工具

Files:

- Create: `src/utils/api.js`

Step 1: 创建API配置和调用函数

创建文件 `src/utils/api.js`：

```
const MODEL_CONFIGS = {
  openai: {
    endpoint: 'https://api.openai.com/v1/chat/completions',
    model: 'gpt-4'
  },
  qianwen: {
    endpoint: 'https://dashscope.aliyuncs.com/api/v1/services/aigc/text-generation/generation',
    model: 'qwen-turbo'
  },
  wenxin: {
    endpoint: 'https://aip.baidubce.com/rpc/2.0/ai_custom/v1/wenxinworkshop/chat/completions',
    model: 'ernie-bot-turbo'
  },
  deepseek: {
    endpoint: 'https://api.deepseek.com/v1/chat/completions',
    model: 'deepseek-chat'
  }
}

const SYSTEM_PROMPT = `你是一个职场沟通专家。我会给你一句带有情绪的"气话"，
请将它转换为5种不同语气的职场表达版本。

要求：
1. 每个版本必须在30字以内
2. 保持礼貌和专业，避免冲突
3. 针对不同场景和对象进行优化

请严格按照以下JSON格式返回，不要添加任何其他文字：
{
  "委婉礼貌": "...",
  "专业正式": "...",
  "友好和谐": "...",
  "幽默风趣": "...",
  "严肃认真": "..."
}`

async function callOpenAI(apiKey, userInput) {
  const config = MODEL_CONFIGS.openai

  const response = await fetch(config.endpoint, {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
```

```
      'Authorization': `Bearer ${apiKey}`
    },
    body: JSON.stringify({
      model: config.model,
      messages: [
        { role: 'system', content: SYSTEM_PROMPT },
        { role: 'user', content: userInput }
      ],
      temperature: 0.7,
      max_tokens: 1000
    })
  })

  if (!response.ok) {
    throw new Error(`OpenAI API error: ${response.status}`)
  }

  const data = await response.json()
  const content = data.choices[0].message.content
  return JSON.parse(content)
}

async function callQianwen(apiKey, userInput) {
  const config = MODEL_CONFIGS.qianwen

  const response = await fetch(config.endpoint, {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
      'Authorization': `Bearer ${apiKey}`
    },
    body: JSON.stringify({
      model: config.model,
      input: {
        messages: [
          { role: 'system', content: SYSTEM_PROMPT },
          { role: 'user', content: userInput }
        ]
      },
      parameters: {
        temperature: 0.7,
        max_tokens: 1000
      }
```

```
    })
  })

  if (!response.ok) {
    throw new Error(`通义千问 API error: ${response.status}`)
  }

  const data = await response.json()
  const content = data.output.text
  return JSON.parse(content)
}

async function callWenxin(apiKey, userInput) {
  const config = MODEL_CONFIGS.wenxin

  const response = await fetch(`${config.endpoint}?access_token=${apiKey}`, {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({
      messages: [
        { role: 'user', content: `${SYSTEM_PROMPT}\n\n用户输入的气话：${userInput}` }
      ],
      temperature: 0.7,
      max_output_tokens: 1000
    })
  })

  if (!response.ok) {
    throw new Error(`文心一言 API error: ${response.status}`)
  }

  const data = await response.json()
  const content = data.result
  return JSON.parse(content)
}

async function callDeepseek(apiKey, userInput) {
  const config = MODEL_CONFIGS.deepseek

  const response = await fetch(config.endpoint, {
    method: 'POST',
```

```
    headers: {
      'Content-Type': 'application/json',
      'Authorization': `Bearer ${apiKey}`
    },
    body: JSON.stringify({
      model: config.model,
      messages: [
        { role: 'system', content: SYSTEM_PROMPT },
        { role: 'user', content: userInput }
      ],
      temperature: 0.7,
      max_tokens: 1000
    })
  })

  if (!response.ok) {
    throw new Error(`Deepseek API error: ${response.status}`)
  }

  const data = await response.json()
  const content = data.choices[0].message.content
  return JSON.parse(content)
}

export async function callAIModel(model, apiKey, userInput) {
  if (!apiKey || !apiKey.trim()) {
    throw new Error('请先在设置中配置API密钥')
  }

  if (!userInput || !userInput.trim()) {
    throw new Error('请输入要转换的内容')
  }

  const callers = {
    openai: callOpenAI,
    qianwen: callQianwen,
    wenxin: callWenxin,
    deepseek: callDeepseek
  }

  const caller = callers[model]
  if (!caller) {
    throw new Error(`不支持的模型: ${model}`)
```

```
  }

  try {
    return await caller(apiKey, userInput)
  } catch (error) {
    // Try to parse error message
    if (error.message.includes('JSON')) {
      throw new Error('AI返回格式错误，请重试')
    }
    throw error
  }
}

export const MODEL_NAMES = {
  openai: 'OpenAI (GPT-4)',
  qianwen: '通义千问',
  wenxin: '文心一言',
  deepseek: 'Deepseek'
}
```

Step 2: 提交API工具

```
git add src/utils/api.js
git commit -m "feat: implement AI API calling utilities"
```

Task 4: 集成API调用到输入组件

Files:

- Modify: `src/components/InputSection.jsx`

Step 1: 更新InputSection集成API调用

修改文件 `src/components/InputSection.jsx`：

```
import { useState } from 'react'
import { useApp } from '../contexts/AppContext'
import { callAIModel } from '../utils/api'
import './InputSection.css'

const EXAMPLES = [
  { text: '这个方案根本行不通！', label: '方案质疑' },
  { text: '你们怎么总是这样？', label: '团队抱怨' },
  { text: '我不干了！', label: '情绪爆发' },
  { text: '这谁写的代码？', label: '代码质疑' },
  { text: '又是你的错！', label: '指责他人' }
]

function InputSection() {
  const {
    selectedModel,
    setSelectedModel,
    apiKeys,
    loading,
    setLoading,
    setError,
    setConversionResult,
    addToHistory
  } = useApp()

  const [input, setInput] = useState('')

  const handleSubmit = async () => {
    if (!input.trim()) return

    const apiKey = apiKeys[selectedModel]
    if (!apiKey) {
      setError('请先在设置中配置API密钥')
      return
    }

    setLoading(true)
    setError(null)

    try {
      const results = await callAIModel(selectedModel, apiKey, input)
      setConversionResult(results)
      addToHistory(input, results, selectedModel)
```

```
  } catch (err) {
    setError(err.message || '转换失败，请重试')
  } finally {
    setLoading(false)
  }
}

const handleKeyDown = (e) => {
  if (e.key === 'Enter' && (e.ctrlKey || e.metaKey)) {
    handleSubmit()
  }
}

return (
  <div className="input-section">
    <h2>输入你的气话</h2>

    <div className="model-selector">
      <label>选择AI模型</label>
      <select
        value={selectedModel}
        onChange={(e) => setSelectedModel(e.target.value)}
      >
        <option value="openai">OpenAI (GPT-4)</option>
        <option value="qianwen">通义千问</option>
        <option value="wenxin">文心一言</option>
        <option value="deepseek">Deepseek</option>
      </select>
    </div>

    <div className="textarea-wrapper">
      <textarea
        value={input}
        onChange={(e) => setInput(e.target.value.slice(0, 500))}
        onKeyDown={handleKeyDown}
        placeholder="在这里输入你想说的气话...（Ctrl+Enter 快速提交）"
        maxLength={500}
      />
      <span className="char-count">{input.length}/500</span>
    </div>

    <div className="button-group">
      <button
```

```
            className="convert-button"
            onClick={handleSubmit}
            disabled={!input.trim() || loading}
          >
            {loading ? (
              <>
                <span className="spinner"></span>
                转换中...
              </>
            ) : (
              '转换'
            )}
          </button>
        </div>

        <div className="examples">
          <h3>试试这些示例：</h3>
          <div className="example-buttons">
            {EXAMPLES.map((example, index) => (
              <button
                key={index}
                className="example-button"
                onClick={() => setInput(example.text)}
              >
                {example.label}
              </button>
            ))}
          </div>
        </div>
      </div>
    )
}

export default InputSection
```

Step 2: 提交更新

```
git add src/components/InputSection.jsx
git commit -m "feat: integrate API calling into InputSection"
```

Task 5: 创建结果卡片组件

Files:

- Create: `src/components/ResultCards.jsx`
- Create: `src/components/ResultCards.css`

Step 1: 创建结果卡片样式

创建文件 `src/components/ResultCards.css`：

```css
.result-cards {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(280px, 1fr));
  gap: 20px;
  margin-bottom: 24px;
}

.result-card {
  background: white;
  border-radius: 12px;
  padding: 20px;
  box-shadow: 0 2px 8px rgba(0, 0, 0, 0.1);
  transition: transform 0.2s, box-shadow 0.2s;
  cursor: pointer;
  position: relative;
}

.result-card:hover {
  transform: translateY(-4px);
  box-shadow: 0 4px 16px rgba(0, 0, 0, 0.15);
}

.card-header {
  display: flex;
  justify-content: space-between;
  align-items: center;
  margin-bottom: 12px;
}

.tone-badge {
  padding: 6px 12px;
  border-radius: 20px;
  font-size: 0.875rem;
  font-weight: 600;
}

.tone-badge.委婉礼貌 {
  background: #e3f2fd;
  color: #1976d2;
}

.tone-badge.专业正式 {
  background: #f3e5f5;
```

```
  color: #7b1fa2;
}

.tone-badge.友好和谐 {
  background: #e8f5e9;
  color: #388e3c;
}

.tone-badge.幽默风趣 {
  background: #fff3e0;
  color: #f57c00;
}

.tone-badge.严肃认真 {
  background: #ffebee;
  color: #d32f2f;
}

.copy-button {
  padding: 6px 12px;
  background: transparent;
  border: 1px solid #e0e0e0;
  border-radius: 6px;
  font-size: 0.875rem;
  cursor: pointer;
  transition: all 0.2s;
  color: #666;
}

.copy-button:hover {
  background: #f5f5f5;
  border-color: #667eea;
  color: #667eea;
}

.copy-button.copied {
  background: #4caf50;
  border-color: #4caf50;
  color: white;
}

.card-content {
  font-size: 1rem;
```

```
  line-height: 1.6;
  color: #333;
  min-height: 60px;
}

.loading-state {
  grid-column: 1 / -1;
  background: white;
  border-radius: 12px;
  padding: 40px;
  text-align: center;
  color: #666;
}

.loading-spinner {
  width: 40px;
  height: 40px;
  margin: 0 auto 16px;
  border: 3px solid #f0f0f0;
  border-top-color: #667eea;
  border-radius: 50%;
  animation: spin 0.8s linear infinite;
}

@keyframes spin {
  to { transform: rotate(360deg); }
}

.error-state {
  grid-column: 1 / -1;
  background: #ffebee;
  border-radius: 12px;
  padding: 20px;
  text-align: center;
  color: #c62828;
}

.empty-state {
  grid-column: 1 / -1;
  background: white;
  border-radius: 12px;
  padding: 40px;
  text-align: center;
```

```
    color: #999;
  }
```

Step 2: 创建结果卡片组件

创建文件 `src/components/ResultCards.jsx`：

```jsx
import { useState } from 'react'
import { useApp } from '../contexts/AppContext'
import './ResultCards.css'

const TONE_STYLES = {
  '委婉礼貌': '委婉礼貌',
  '专业正式': '专业正式',
  '友好和谐': '友好和谐',
  '幽默风趣': '幽默风趣',
  '严肃认真': '严肃认真'
}

function ResultCards() {
  const { conversionResult, loading, error } = useApp()
  const [copiedCard, setCopiedCard] = useState(null)

  const handleCopy = async (tone, text, e) => {
    e.stopPropagation()
    try {
      await navigator.clipboard.writeText(text)
      setCopiedCard(tone)
      setTimeout(() => setCopiedCard(null), 2000)
    } catch (err) {
      console.error('复制失败:', err)
    }
  }

  if (loading) {
    return (
      <div className="result-cards">
        <div className="loading-state">
          <div className="loading-spinner"></div>
          <p>AI正在转换中，请稍候...</p>
        </div>
      </div>
    )
  }

  if (error) {
    return (
      <div className="result-cards">
        <div className="error-state">
          <p>{error}</p>
```

```
        </div>
      </div>
    )
  }

  if (!conversionResult) {
    return (
      <div className="result-cards">
        <div className="empty-state">
          <p>输入气话并点击"转换"按钮，即可看到5种不同语气的职场表达</p>
        </div>
      </div>
    )
  }

  return (
    <div className="result-cards">
      {Object.entries(conversionResult).map(([tone, text]) => (
        <div
          key={tone}
          className="result-card"
          onClick={(e) => handleCopy(tone, text, e)}
        >
          <div className="card-header">
            <span className={`tone-badge ${TONE_STYLES[tone]}`}>
              {tone}
            </span>
            <button
              className={`copy-button ${copiedCard === tone ? 'copied' : ''}`}
              onClick={(e) => handleCopy(tone, text, e)}
            >
              {copiedCard === tone ? '已复制' : '复制'}
            </button>
          </div>
          <div className="card-content">
            {text}
          </div>
        </div>
      ))}
    </div>
  )
}
```

```
export default ResultCards
```

Step 3: 在App.jsx中使用ResultCards

修改文件 `src/App.jsx`：

```
import { AppProvider } from './contexts/AppContext'
import InputSection from './components/InputSection'
import ResultCards from './components/ResultCards'
import './App.css'

function App() {
  return (
    <AppProvider>
      <div className="app">
        <h1>职场说话保命神器</h1>
        <InputSection />
        <ResultCards />
      </div>
    </AppProvider>
  )
}

export default App
```

Step 4: 测试结果卡片

```
npm run dev
```

Expected: 输入文字并点击转换后，看到5个卡片显示不同语气的转换结果

Step 5: 提交结果卡片

```
git add src/components/ResultCards.jsx src/components/ResultCards.css src/App.jsx
git commit -m "feat: implement ResultCards component"
```

Task 6: 创建历史记录组件

Files:

- Create: `src/components/HistorySection.jsx`
- Create: `src/components/HistorySection.css`

Step 1: 创建历史记录样式

创建文件 `src/components/HistorySection.css`：

```css
.history-section {
  background: white;
  border-radius: 12px;
  padding: 24px;
  box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
}

.history-header {
  display: flex;
  justify-content: space-between;
  align-items: center;
  margin-bottom: 20px;
}

.history-header h2 {
  font-size: 1.25rem;
  color: #333;
}

.clear-button {
  padding: 8px 16px;
  background: #f5f5f5;
  border: 1px solid #e0e0e0;
  border-radius: 6px;
  font-size: 0.875rem;
  cursor: pointer;
  transition: all 0.2s;
}

.clear-button:hover {
  background: #ffebee;
  border-color: #ef5350;
  color: #c62828;
}

.history-list {
  display: flex;
  flex-direction: column;
  gap: 16px;
  max-height: 400px;
  overflow-y: auto;
}
```

```css
.history-item {
  padding: 16px;
  background: #f9f9f9;
  border-radius: 8px;
  cursor: pointer;
  transition: background 0.2s;
}

.history-item:hover {
  background: #f0f0f0;
}

.history-input {
  font-weight: 600;
  color: #333;
  margin-bottom: 8px;
}

.history-results {
  display: flex;
  flex-wrap: wrap;
  gap: 8px;
}

.history-result {
  padding: 4px 8px;
  background: white;
  border-radius: 4px;
  font-size: 0.875rem;
  color: #666;
}

.history-meta {
  display: flex;
  justify-content: space-between;
  align-items: center;
  margin-top: 8px;
  font-size: 0.875rem;
  color: #999;
}

.empty-history {
  text-align: center;
```

```
  padding: 40px;
  color: #999;
}
```

Step 2: 创建历史记录组件

创建文件 `src/components/HistorySection.jsx`：

```jsx
import { useApp } from '../contexts/AppContext'
import { MODEL_NAMES } from '../utils/api'
import './HistorySection.css'

function HistorySection() {
  const { history, clearHistory, setConversionResult } = useApp()

  const formatTimestamp = (timestamp) => {
    const date = new Date(timestamp)
    const now = new Date()
    const diff = now - date

    if (diff < 60000) return '刚刚'
    if (diff < 3600000) return `${Math.floor(diff / 60000)}分钟前`
    if (diff < 86400000) return `${Math.floor(diff / 3600000)}小时前`
    return date.toLocaleDateString('zh-CN')
  }

  const handleItemClick = (item) => {
    setConversionResult(item.results)
  }

  if (history.length === 0) {
    return (
      <div className="history-section">
        <div className="history-header">
          <h2>历史记录</h2>
        </div>
        <div className="empty-history">
          暂无历史记录
        </div>
      </div>
    )
  }

  return (
    <div className="history-section">
      <div className="history-header">
        <h2>历史记录 ({history.length})</h2>
        <button className="clear-button" onClick={clearHistory}>
          清除历史
        </button>
      </div>
```

```
      <div className="history-list">
        {history.map((item) => (
          <div
            key={item.id}
            className="history-item"
            onClick={() => handleItemClick(item)}
          >
            <div className="history-input">
              {item.input}
            </div>
            <div className="history-results">
              {Object.entries(item.results).map(([tone, text]) => (
                <span key={tone} className="history-result">
                  {tone}: {text}
                </span>
              ))}
            </div>
            <div className="history-meta">
              <span>{MODEL_NAMES[item.model]}</span>
              <span>{formatTimestamp(item.timestamp)}</span>
            </div>
          </div>
        ))}
      </div>
    </div>
  )
}

export default HistorySection
```

Step 3: 在App.jsx中使用历史记录

修改文件 `src/App.jsx`：

```jsx
import { AppProvider } from './contexts/AppContext'
import InputSection from './components/InputSection'
import ResultCards from './components/ResultCards'
import HistorySection from './components/HistorySection'
import './App.css'

function App() {
  return (
    <AppProvider>
      <div className="app">
        <h1>职场说话保命神器</h1>
        <InputSection />
        <ResultCards />
        <HistorySection />
      </div>
    </AppProvider>
  )
}

export default App
```

Step 4: 测试历史记录

```bash
npm run dev
```

Expected: 转换后，历史记录中显示之前的转换结果，点击可重新查看

Step 5: 提交历史记录

```bash
git add src/components/HistorySection.jsx src/components/HistorySection.css src/App.jsx
git commit -m "feat: implement HistorySection component"
```

Task 7: 创建设置组件

Files:

- Create: `src/components/SettingsPanel.jsx`
- Create: `src/components/SettingsPanel.css`

Step 1: 创建设置面板样式

创建文件 `src/components/SettingsPanel.css`：

```css
.settings-panel {
  background: white;
  border-radius: 12px;
  padding: 24px;
  box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
}

.settings-header {
  margin-bottom: 24px;
}

.settings-header h2 {
  font-size: 1.25rem;
  color: #333;
  margin-bottom: 8px;
}

.settings-header p {
  font-size: 0.875rem;
  color: #666;
}

.api-key-section {
  margin-bottom: 24px;
}

.api-key-section h3 {
  font-size: 1rem;
  color: #444;
  margin-bottom: 12px;
}

.api-key-item {
  margin-bottom: 16px;
}

.api-key-label {
  display: block;
  font-weight: 600;
  color: #555;
  margin-bottom: 6px;
  font-size: 0.875rem;
}
```

```css
.api-key-input {
  width: 100%;
  padding: 10px 12px;
  border: 2px solid #e0e0e0;
  border-radius: 8px;
  font-size: 0.875rem;
  transition: border-color 0.2s;
}

.api-key-input:focus {
  outline: none;
  border-color: #667eea;
}

.api-key-input::placeholder {
  color: #bbb;
}

.security-notice {
  background: #fff3e0;
  border-left: 4px solid #ff9800;
  padding: 12px 16px;
  border-radius: 4px;
  font-size: 0.875rem;
  color: #e65100;
  margin-top: 16px;
}

.security-notice strong {
  display: block;
  margin-bottom: 4px;
}

.api-link {
  color: #667eea;
  text-decoration: none;
  font-size: 0.875rem;
}

.api-link:hover {
  text-decoration: underline;
}
```

Step 2: 创建设置面板组件

创建文件 `src/components/SettingsPanel.jsx`：

```jsx
import { useApp } from '../contexts/AppContext'
import './SettingsPanel.css'

const API_INFO = {
  openai: {
    name: 'OpenAI API Key',
    placeholder: 'sk-...',
    link: 'https://platform.openai.com/api-keys'
  },
  qianwen: {
    name: '通义千问 API Key',
    placeholder: 'sk-...',
    link: 'https://dashscope.aliyun.com/apiKey'
  },
  wenxin: {
    name: '文心一言 Access Token',
    placeholder: '输入您的Access Token',
    link: 'https://cloud.baidu.com/product/wenxinworkshop'
  },
  deepseek: {
    name: 'Deepseek API Key',
    placeholder: 'sk-...',
    link: 'https://platform.deepseek.com/api_keys'
  }
}

function SettingsPanel() {
  const { apiKeys, updateApiKey } = useApp()

  const handleKeyChange = (model, value) => {
    updateApiKey(model, value)
  }

  return (
    <div className="settings-panel">
      <div className="settings-header">
        <h2>API 密钥设置</h2>
        <p>配置您的AI模型API密钥，密钥仅保存在浏览器本地</p>
      </div>

      {Object.entries(API_INFO).map(([model, info]) => (
        <div key={model} className="api-key-section">
          <h3>{info.name}</h3>
```

```
            <div className="api-key-item">
              <label className="api-key-label">API 密钥</label>
              <input
                type="password"
                className="api-key-input"
                placeholder={info.placeholder}
                value={apiKeys[model] || ''}
                onChange={(e) => handleKeyChange(model, e.target.value)}
              />
              <a
                href={info.link}
                target="_blank"
                rel="noopener noreferrer"
                className="api-link"
              >
                获取API密钥 →
              </a>
            </div>
          </div>
        ))}

        <div className="security-notice">
          <strong>安全提示：</strong>
          您的API密钥仅存储在浏览器本地，不会上传到任何服务器。请妥善保管您的密钥，不要分享给他人。
        </div>
      </div>
    )
}

export default SettingsPanel
```

Step 3: 创建容器组件支持切换视图

修改文件 `src/App.jsx`：

```jsx
import { useState } from 'react'
import { AppProvider, useApp } from './contexts/AppContext'
import InputSection from './components/InputSection'
import ResultCards from './components/ResultCards'
import HistorySection from './components/HistorySection'
import SettingsPanel from './components/SettingsPanel'
import './App.css'

function AppContent() {
  const [activeTab, setActiveTab] = useState('home')
  const { error } = useApp()

  return (
    <div className="app">
      <h1>职场说话保命神器</h1>

      <div className="tabs">
        <button
          className={`tab ${activeTab === 'home' ? 'active' : ''}`}
          onClick={() => setActiveTab('home')}
        >
          转换
        </button>
        <button
          className={`tab ${activeTab === 'settings' ? 'active' : ''}`}
          onClick={() => setActiveTab('settings')}
        >
          设置
        </button>
      </div>

      {activeTab === 'home' && (
        <>
          <InputSection />
          {error && <div className="error-message">{error}</div>}
          <ResultCards />
          <HistorySection />
        </>
      )}

      {activeTab === 'settings' && (
        <SettingsPanel />
      )}
```

```
      </div>
    )
  }

  function App() {
    return (
      <AppProvider>
        <AppContent />
      </AppProvider>
    )
  }

  export default App
```

Step 4: 添加标签页样式

修改文件 `src/App.css`：

```css
/* 在文件末尾添加 */

.tabs {
  display: flex;
  gap: 12px;
  margin-bottom: 24px;
  justify-content: center;
}

.tab {
  padding: 12px 24px;
  background: white;
  border: none;
  border-radius: 8px;
  font-size: 1rem;
  font-weight: 600;
  cursor: pointer;
  transition: all 0.2s;
  color: #666;
}

.tab:hover {
  background: #f5f5f5;
}

.tab.active {
  background: white;
  color: #667eea;
  box-shadow: 0 2px 8px rgba(102, 126, 234, 0.3);
}

.error-message {
  background: #ffebee;
  color: #c62828;
  padding: 12px 16px;
  border-radius: 8px;
  margin-bottom: 20px;
  text-align: center;
}
```

Step 5: 测试设置面板

```
npm run dev
```

Expected: 可以切换到设置标签页，配置各模型的API密钥

Step 6: 提交设置面板

```
git add src/components/SettingsPanel.jsx src/components/SettingsPanel.css src/App.jsx src/App.cs
git commit -m "feat: implement SettingsPanel with tab navigation"
```

Task 8: 响应式设计优化

Files:

- Modify: `src/App.css`

Step 1: 添加响应式样式

修改文件 `src/App.css`：

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

body {
  font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto', sans-serif;
  background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
  min-height: 100vh;
}

.app {
  max-width: 1200px;
  margin: 0 auto;
  padding: 20px;
}

h1 {
  text-align: center;
  color: white;
  margin-bottom: 40px;
  font-size: 2.5rem;
}

/* 响应式设计 */

@media (max-width: 768px) {
  .app {
    padding: 12px;
  }

  h1 {
    font-size: 1.75rem;
    margin-bottom: 24px;
  }

  .tabs {
    flex-direction: row;
    gap: 8px;
  }

  .tab {
```

```
    padding: 10px 16px;
    font-size: 0.875rem;
  }
}

@media (max-width: 480px) {
  h1 {
    font-size: 1.5rem;
  }

  .tabs {
    flex-direction: column;
  }

  .tab {
    width: 100%;
  }
}

/* 标签页样式 */

.tabs {
  display: flex;
  gap: 12px;
  margin-bottom: 24px;
  justify-content: center;
}

.tab {
  padding: 12px 24px;
  background: white;
  border: none;
  border-radius: 8px;
  font-size: 1rem;
  font-weight: 600;
  cursor: pointer;
  transition: all 0.2s;
  color: #666;
}

.tab:hover {
  background: #f5f5f5;
}
```

```css
.tab.active {
  background: white;
  color: #667eea;
  box-shadow: 0 2px 8px rgba(102, 126, 234, 0.3);
}

.error-message {
  background: #ffebee;
  color: #c62828;
  padding: 12px 16px;
  border-radius: 8px;
  margin-bottom: 20px;
  text-align: center;
}
```

Step 2: 添加响应式卡片样式

修改文件 `src/components/ResultCards.css`：

```css
/* 在文件末尾添加 */

@media (max-width: 768px) {
  .result-cards {
    grid-template-columns: 1fr;
    gap: 16px;
  }

  .result-card {
    padding: 16px;
  }
}

@media (max-width: 480px) {
  .result-card {
    padding: 12px;
  }

  .card-content {
    font-size: 0.9375rem;
  }
}
```

Step 3: 提交响应式设计

```
git add src/App.css src/components/ResultCards.css
git commit -m "feat: add responsive design for mobile devices"
```

Task 9: 添加图标优化

Files:

- Modify: `package.json`
- Modify: `src/components/SettingsPanel.jsx`

Step 1: 安装图标库

```
npm install lucide-react
```

Step 2: 更新设置面板使用图标

修改文件 `src/components/SettingsPanel.jsx`：

```jsx
import { useApp } from '../contexts/AppContext'
import { ExternalLink } from 'lucide-react'
import './SettingsPanel.css'

const API_INFO = {
  openai: {
    name: 'OpenAI API Key',
    placeholder: 'sk-...',
    link: 'https://platform.openai.com/api-keys'
  },
  qianwen: {
    name: '通义千问 API Key',
    placeholder: 'sk-...',
    link: 'https://dashscope.aliyun.com/apiKey'
  },
  wenxin: {
    name: '文心一言 Access Token',
    placeholder: '输入您的Access Token',
    link: 'https://cloud.baidu.com/product/wenxinworkshop'
  },
  deepseek: {
    name: 'Deepseek API Key',
    placeholder: 'sk-...',
    link: 'https://platform.deepseek.com/api_keys'
  }
}

function SettingsPanel() {
  const { apiKeys, updateApiKey } = useApp()

  const handleKeyChange = (model, value) => {
    updateApiKey(model, value)
  }

  return (
    <div className="settings-panel">
      <div className="settings-header">
        <h2>API 密钥设置</h2>
        <p>配置您的AI模型API密钥，密钥仅保存在浏览器本地</p>
      </div>

      {Object.entries(API_INFO).map(([model, info]) => (
        <div key={model} className="api-key-section">
```

```
            <h3>{info.name}</h3>
            <div className="api-key-item">
              <label className="api-key-label">API 密钥</label>
              <input
                type="password"
                className="api-key-input"
                placeholder={info.placeholder}
                value={apiKeys[model] || ''}
                onChange={(e) => handleKeyChange(model, e.target.value)}
              />
              <a
                href={info.link}
                target="_blank"
                rel="noopener noreferrer"
                className="api-link"
              >
                <ExternalLink size={14} style={{ verticalAlign: 'middle', marginRight: 4 }} />
                获取API密钥 →
              </a>
            </div>
          </div>
        ))}

        <div className="security-notice">
          <strong>安全提示：</strong>
          您的API密钥仅存储在浏览器本地，不会上传到任何服务器。请妥善保管您的密钥，不要分享给他人。
        </div>
      </div>
    )
}

export default SettingsPanel
```

Step 3: 提交图标更新

```
git add package.json package-lock.json src/components/SettingsPanel.jsx
git commit -m "feat: add lucide-react icons for better UX"
```

构建与部署准备

Task 10: 生产构建测试

Step 1: 构建生产版本

```
npm run build
```

Expected: 生成 `dist` 目录，包含优化后的静态文件

Step 2: 预览构建结果

```
npm run preview
```

Expected: 在预览服务器中查看生产构建版本

Step 3: 提交构建配置

```
git add .
git commit -m "chore: complete implementation and test build"
```

最终检查清单

- [ ] 所有组件正常工作
- [ ] API密钥保存功能正常
- [ ] 历史记录功能正常
- [ ] 响应式设计在不同设备上正常
- [ ] 复制功能正常
- [ ] 示例按钮功能正常
- [ ] 错误处理友好
- [ ] 生产构建成功

实施完成后的下一步

1. **部署到Vercel**（推荐）

   ```
   npm install -g vercel
   vercel
   ```

2. **或部署到其他平台**（Netlify、GitHub Pages等）
3. **测试实际API调用**（需要配置API密钥）
4. **根据实际使用情况进行优化**

计划总任务数： 10
预计实施时间： 2-3小时
技术难度： 中等

注意事项：

- 按顺序完成每个任务
- 每个Step都是独立的，完成后立即提交
- 遇到问题及时调试，不要积累问题
- 使用浏览器开发工具查看console和network请求
- 确保每个功能点都有对应的错误处理

参考资源：

- React文档
- Vite文档
- localStorage MDN
- 各大模型API文档