FlashGenius: AI-Powered Flashcard Generator

FlashGenius

Generate intelligent flashcards with AI

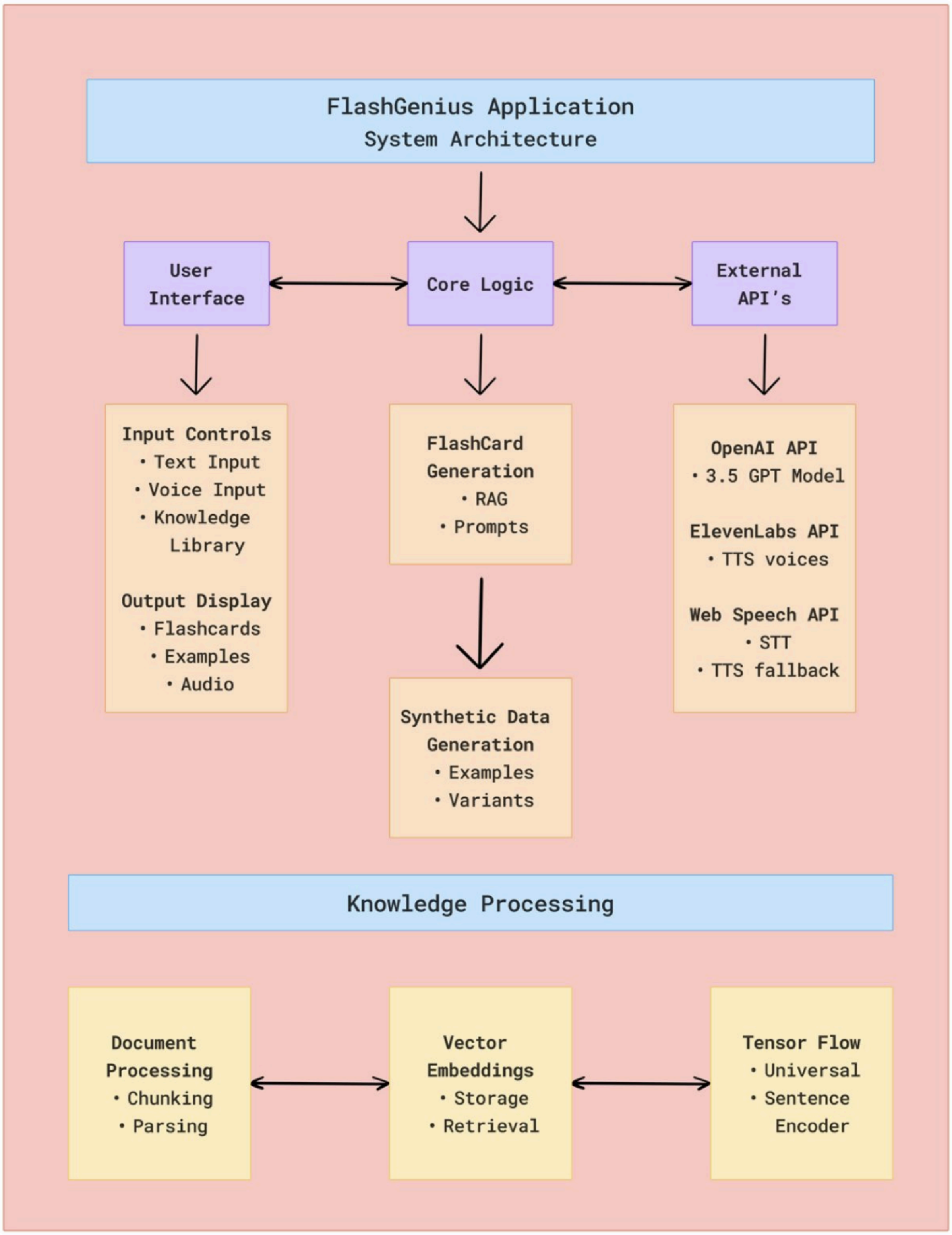
Comprehensive Project Documentation

Introduction

FlashGenius is an innovative AI-powered flashcard generator that enhances the learning experience by implementing four key generative AI technologies from our project requirements. We successfully integrated **Prompt Engineering** techniques with structured prompts that adapt to different difficulty levels, ensuring consistent, high-quality flashcard content. Our implementation of **Retrieval-Augmented Generation** (RAG) processes user-uploaded documents through TensorFlow.js and Universal Sentence Encoder, creating vector embeddings for semantic search to ground flashcards in reliable source material with proper citations. This approach ensures the information is accurate and directly relevant to the user's study materials.

Additionally, we implemented **Multimodal Integration** by combining text with speech recognition and text-to-speech capabilities, accommodating different learning styles and improving accessibility. Users can input topics verbally and listen to flashcard content through high-quality voice synthesis. Our **Synthetic Data Generation** feature creates alternative examples and real-world applications of concepts, helping learners understand topics from multiple perspectives and different contexts. All these features work together in a privacy-conscious manner, with client-side processing ensuring user data remains secure while still delivering a powerful, adaptive learning tool that transforms how students create and interact with study materials.

1. System Architecture



FlashGenius utilizes a modular architecture organized into interconnected components that work together to deliver a comprehensive flashcard generation experience. At the highest level, the application consists of three primary layers: the User Interface, Core Logic, and External APIs, all working in concert to process and present educational content.

The User Interface layer handles all user interactions and is divided into Input Controls and Output Display components. The Input Controls manage text input for topics or materials, voice input through speech recognition, and the knowledge library interface where users upload and manage study documents. The Output Display component presents flashcards with interactive flip functionality, examples generated through synthetic data, and audio playback for voice output.

The Core Logic layer contains the intelligence of the system, with the Flashcard Generation module handling prompt creation and RAG integration, while the Synthetic Data Generation module creates real-world examples and contextual variations. These components implement the prompt engineering and synthetic data generation requirements by crafting specialized prompts and creating additional learning materials based on user content.

External APIs connect FlashGenius to powerful AI services, including the OpenAI API for GPT model interactions that generate flashcards and examples, the ElevenLabs API for high-quality text-to-speech conversion, and the Web Speech API for browser-based speech recognition and synthesis fallback. These integrations enable the multimodal integration requirement by combining text with speech input and output capabilities.

The Knowledge Processing layer forms the foundation of the RAG implementation, consisting of Document Processing functions that chunk and parse uploaded materials, Vector Embeddings that create and store semantic representations using TensorFlow.js and the Universal Sentence Encoder, and the retrieval mechanisms that find relevant content for flashcard generation.

All these components operate primarily client-side, maintaining privacy while still delivering advanced AI capabilities. The architecture emphasizes modularity, allowing components to be improved or replaced independently while maintaining overall system functionality. This thoughtful organization enables FlashGenius to meet all four implemented requirements (Prompt Engineering, RAG, Multimodal Integration, and Synthetic Data Generation) in a cohesive, user-centered application.

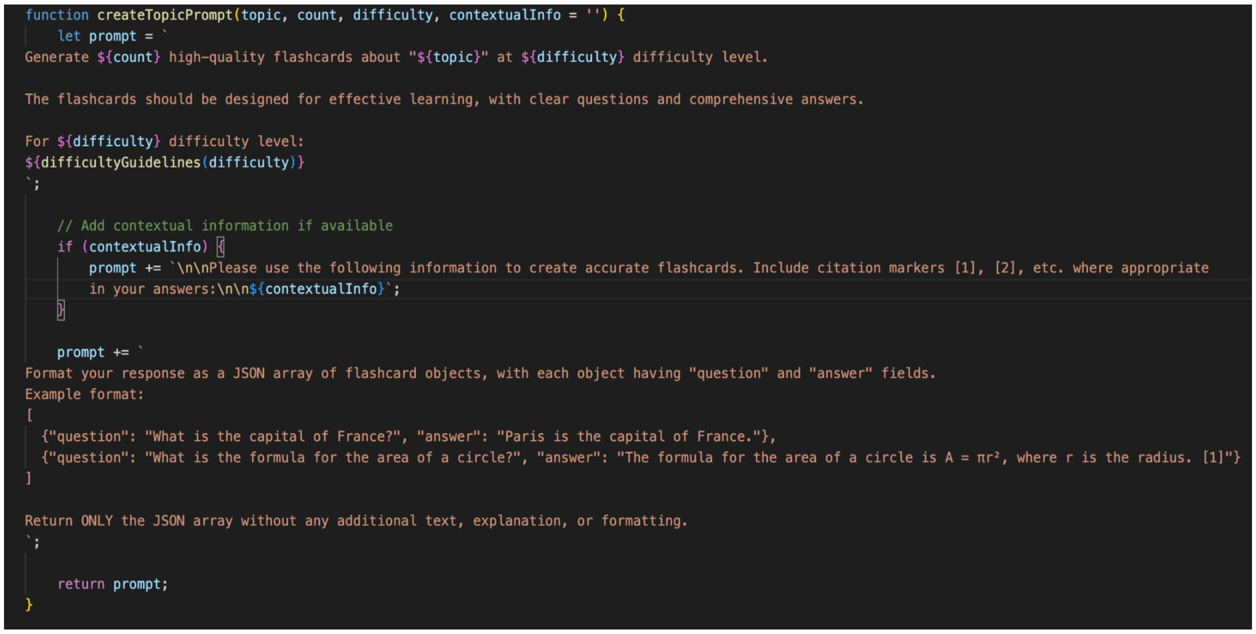
2. Implementation Details-Core Components Implemented

2.1 Prompt Engineering

FlashGenius implements sophisticated prompt engineering techniques to generate high-quality educational content:

- **Contextual Prompts**: Dynamic prompt construction based on user input, chosen difficulty level, and knowledge base context
- **Multi-part Instructions**: Structured prompts with clear formatting requirements for consistent output
- **Format Control**: JSON output specification ensuring properly structured flashcard data
- **Difficulty Calibration**: Tailored prompting strategies for basic, intermediate, and advanced difficulty levels
- **Error Handling**: Robust fallback mechanisms when API responses don't meet expected formats

Example prompt structure for flashcard generation:



```
function createTopicPrompt(topic, count, difficulty, contextualInfo = '') {
    let prompt = `
Generate ${count} high-quality flashcards about "${topic}" at ${difficulty} difficulty level.

The flashcards should be designed for effective learning, with clear questions and comprehensive answers.

For ${difficulty} difficulty level:
${difficultyGuidelines(difficulty)}
`;

    // Add contextual information if available
    if (contextualInfo) {
        prompt += `\n\nPlease use the following information to create accurate flashcards. Include citation markers [1], [2], etc. where appropriate
        in your answers:\n\n${contextualInfo}`;
    }

    prompt += `
Format your response as a JSON array of flashcard objects, with each object having "question" and "answer" fields.
Example format:
[
  {"question": "What is the capital of France?", "answer": "Paris is the capital of France."},
  {"question": "What is the formula for the area of a circle?", "answer": "The formula for the area of a circle is A = πr², where r is the radius. [1]"}
]

Return ONLY the JSON array without any additional text, explanation, or formatting.
`;

    return prompt;
}
```

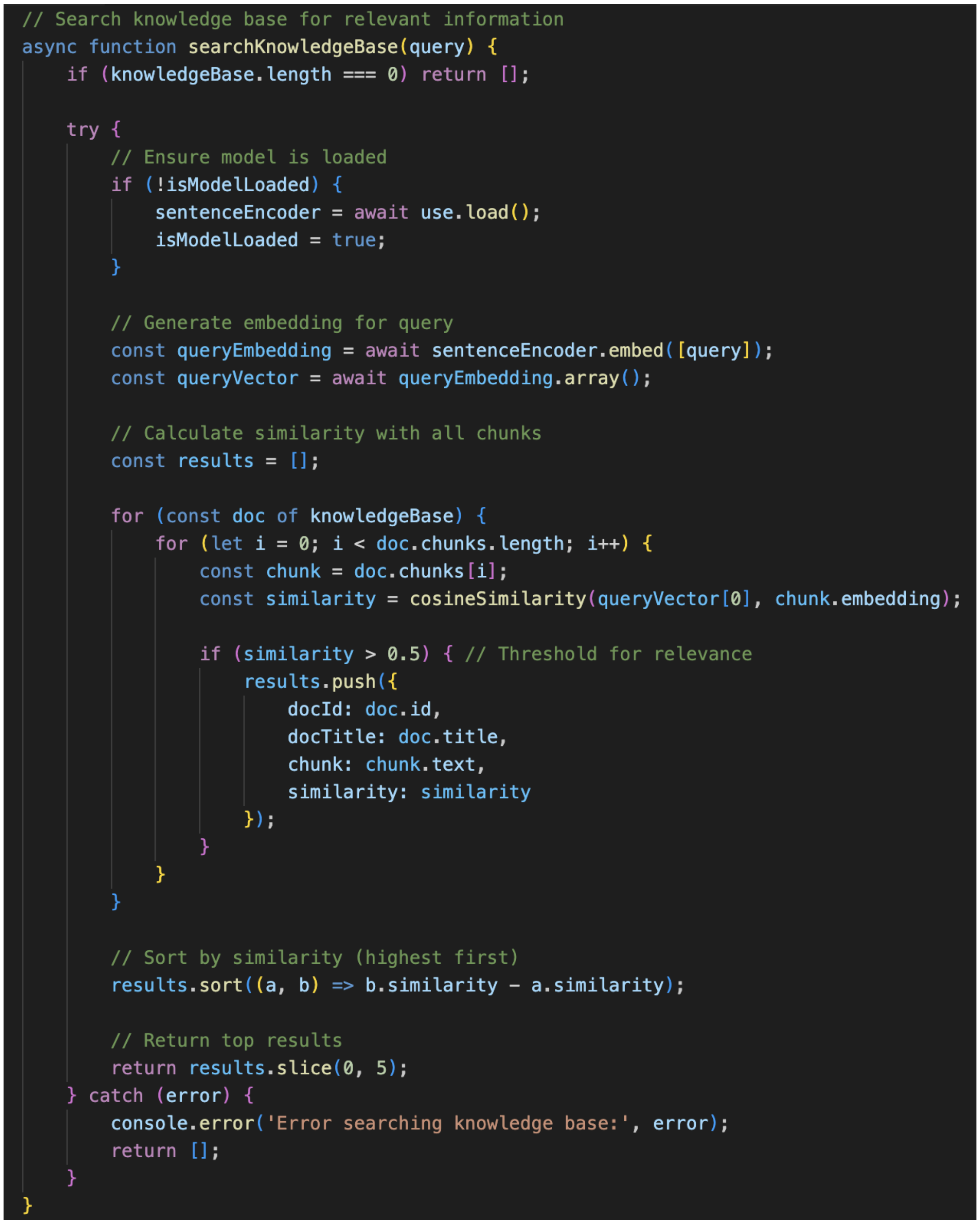
2.2 Retrieval-Augmented Generation (RAG)

FlashGenius implements a client-side RAG system for personalized flashcard generation:

- **Knowledge Base**: Client-side storage of user-uploaded documents
- **Document Processing**: Parsing and chunking text and PDF documents

- **Vector Embeddings**: TensorFlow.js with Universal Sentence Encoder for semantic embeddings
- **Similarity Search**: Cosine similarity for retrieving relevant content
- **Context Integration**: Incorporating retrieved context into flashcard generation
- **Citation Tracking**: Maintaining source references in generated content

Key RAG implementation:



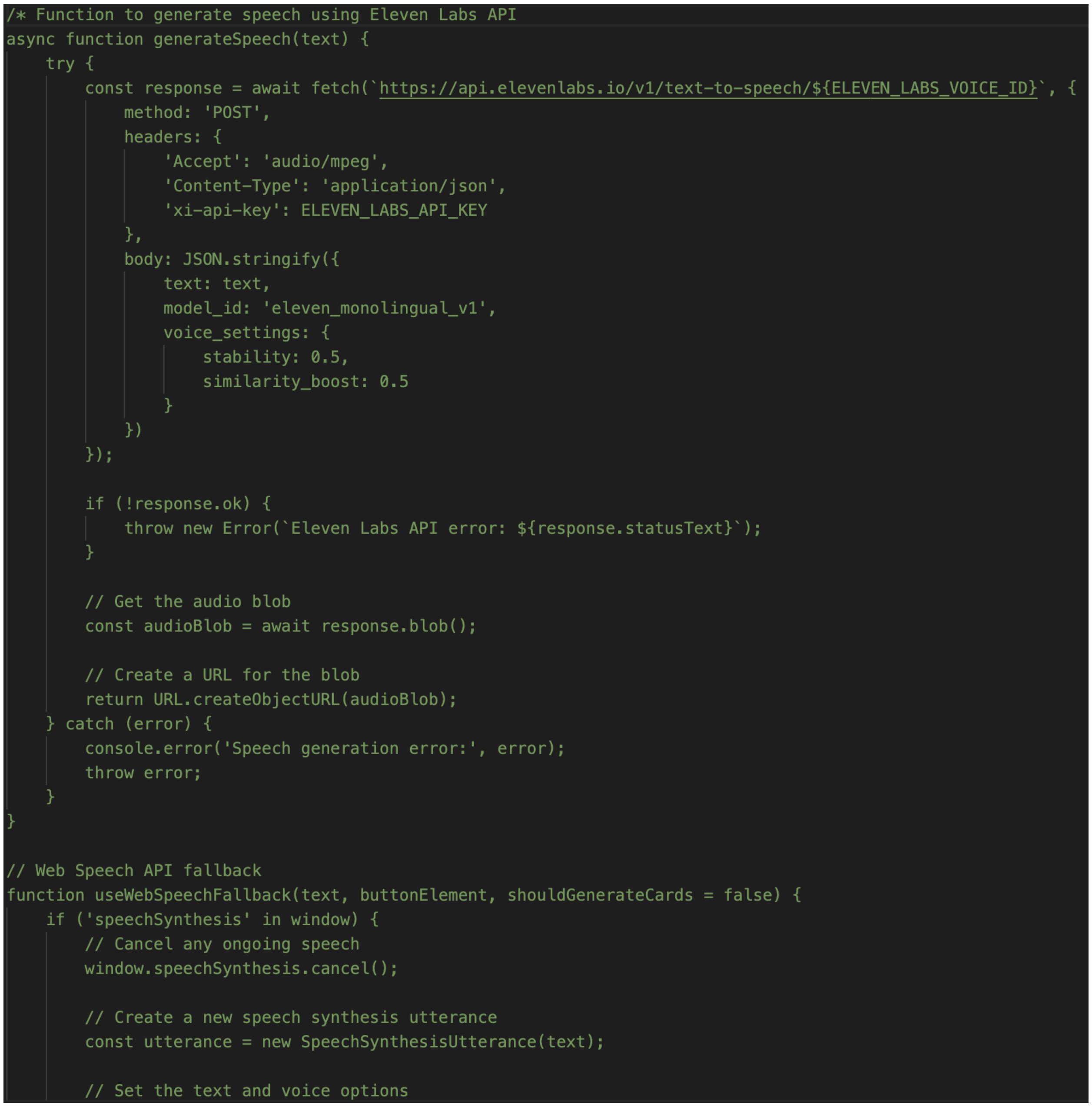
```javascript
// Search knowledge base for relevant information
async function searchKnowledgeBase(query) {
    if (knowledgeBase.length === 0) return [];

    try {
        // Ensure model is loaded
        if (!isModelLoaded) {
            sentenceEncoder = await use.load();
            isModelLoaded = true;
        }

        // Generate embedding for query
        const queryEmbedding = await sentenceEncoder.embed([query]);
        const queryVector = await queryEmbedding.array();

        // Calculate similarity with all chunks
        const results = [];

        for (const doc of knowledgeBase) {
            for (let i = 0; i < doc.chunks.length; i++) {
                const chunk = doc.chunks[i];
                const similarity = cosineSimilarity(queryVector[0], chunk.embedding);

                if (similarity > 0.5) { // Threshold for relevance
                    results.push({
                        docId: doc.id,
                        docTitle: doc.title,
                        chunk: chunk.text,
                        similarity: similarity
                    });
                }
            }
        }

        // Sort by similarity (highest first)
        results.sort((a, b) => b.similarity - a.similarity);

        // Return top results
        return results.slice(0, 5);
    } catch (error) {
        console.error('Error searching knowledge base:', error);
        return [];
    }
}
```

2.3 Multimodal Integration

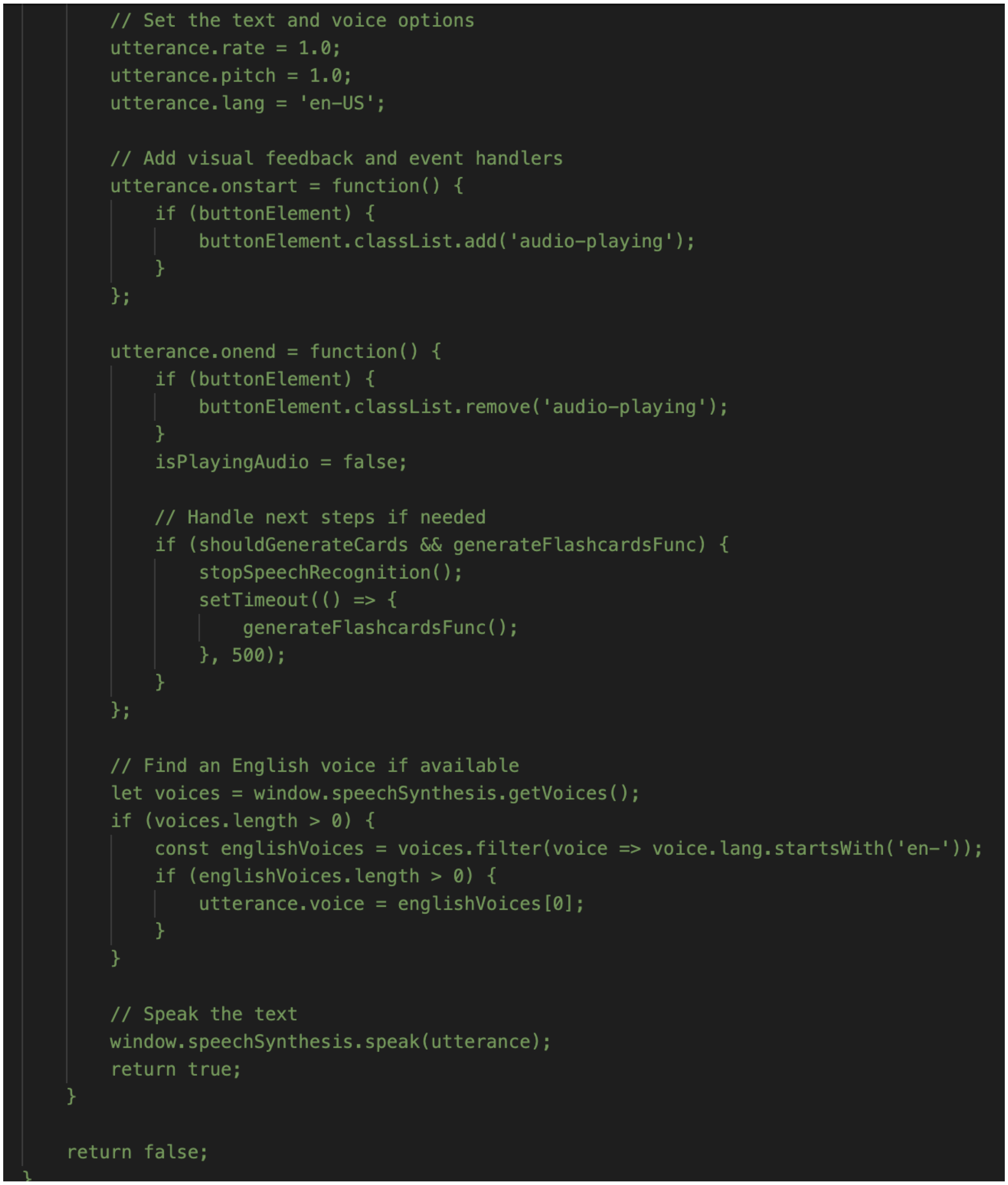
FlashGenius combines text generation with speech input and output:

- **Text-to-Speech**: Integration with ElevenLabs API for high-quality voice output
- **Speech-to-Text**: Web Speech API for voice input
- **Fallback Mechanisms**: Web Speech API as TTS fallback
- **Cross-Modal Interaction**: Seamless transition between text and speech modalities
- **User Control**: Toggle mechanisms for audio playback

Implementation highlights:



```
/* Function to generate speech using Eleven Labs API
async function generateSpeech(text) {
    try {
        const response = await fetch(`https://api.elevenlabs.io/v1/text-to-speech/${ELEVEN_LABS_VOICE_ID}`, {
            method: 'POST',
            headers: {
                'Accept': 'audio/mpeg',
                'Content-Type': 'application/json',
                'xi-api-key': ELEVEN_LABS_API_KEY
            },
            body: JSON.stringify({
                text: text,
                model_id: 'eleven_monolingual_v1',
                voice_settings: {
                    stability: 0.5,
                    similarity_boost: 0.5
                }
            })
        });

        if (!response.ok) {
            throw new Error(`Eleven Labs API error: ${response.statusText}`);
        }

        // Get the audio blob
        const audioBlob = await response.blob();

        // Create a URL for the blob
        return URL.createObjectURL(audioBlob);
    } catch (error) {
        console.error('Speech generation error:', error);
        throw error;
    }
}

// Web Speech API fallback
function useWebSpeechFallback(text, buttonElement, shouldGenerateCards = false) {
    if ('speechSynthesis' in window) {
        // Cancel any ongoing speech
        window.speechSynthesis.cancel();

        // Create a new speech synthesis utterance
        const utterance = new SpeechSynthesisUtterance(text);

        // Set the text and voice options
```



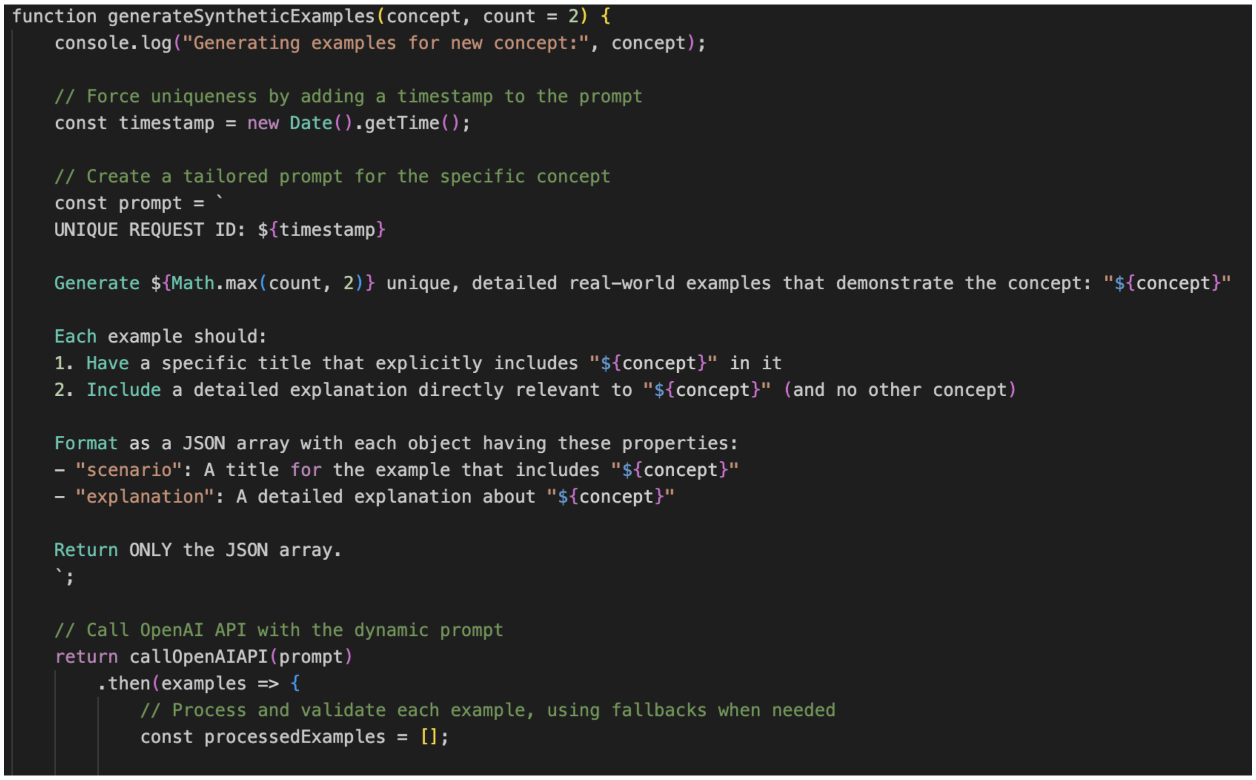
```
        // Set the text and voice options
        utterance.rate = 1.0;
        utterance.pitch = 1.0;
        utterance.lang = 'en-US';

        // Add visual feedback and event handlers
        utterance.onstart = function() {
            if (buttonElement) {
                buttonElement.classList.add('audio-playing');
            }
        };

        utterance.onend = function() {
            if (buttonElement) {
                buttonElement.classList.remove('audio-playing');
            }
            isPlayingAudio = false;

            // Handle next steps if needed
            if (shouldGenerateCards && generateFlashcardsFunc) {
                stopSpeechRecognition();
                setTimeout(() => {
                    generateFlashcardsFunc();
                }, 500);
            }
        };

        // Find an English voice if available
        let voices = window.speechSynthesis.getVoices();
        if (voices.length > 0) {
            const englishVoices = voices.filter(voice => voice.lang.startsWith('en-'));
            if (englishVoices.length > 0) {
                utterance.voice = englishVoices[0];
            }
        }

        // Speak the text
        window.speechSynthesis.speak(utterance);
        return true;
    }

    return false;
}
```

2.4 Synthetic Data Generation

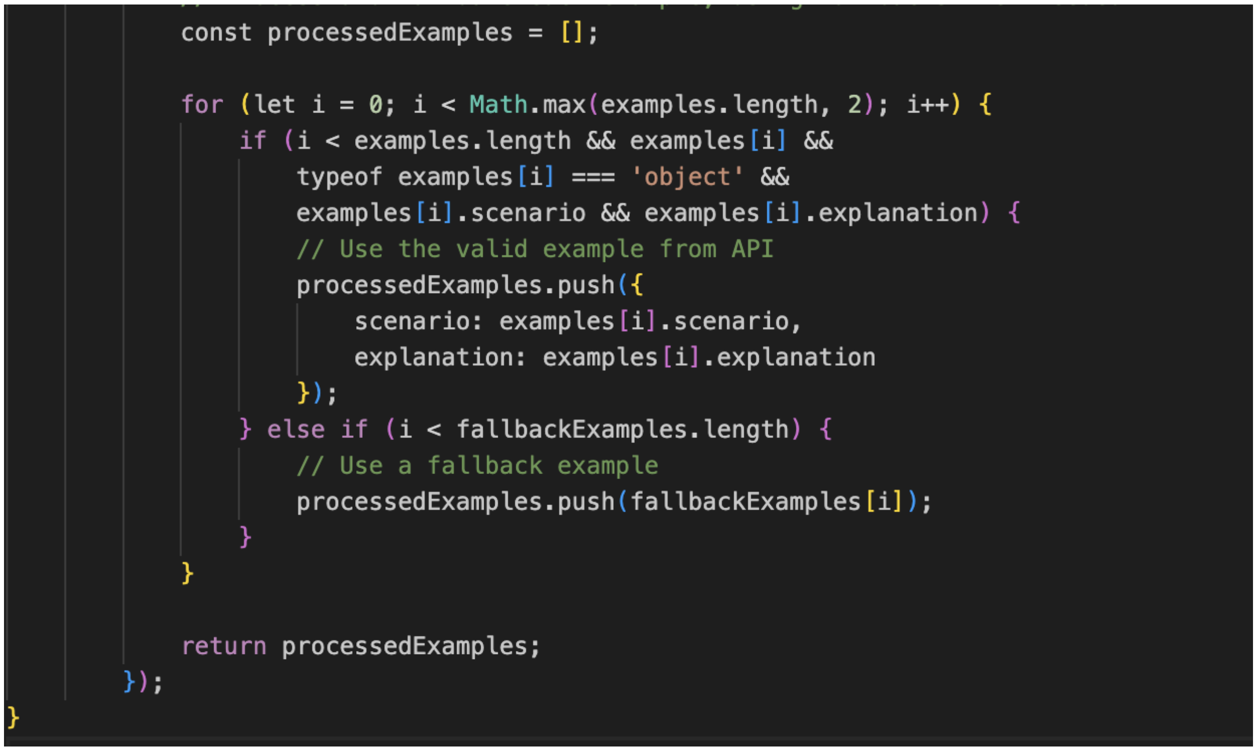
FlashGenius creates additional, contextually relevant learning materials:

- **Example Generation**: Creates real-world examples for concepts
- **Multiple Perspectives**: Presents concepts from different angles
- **Domain-Specific Contexts**: Generates examples in various fields (healthcare, technology, education)
- **Fallback Mechanisms**: Ensures quality even when API responses are suboptimal
- **User Control**: Toggle for enabling/disabling synthetic data generation

Implementation highlights:



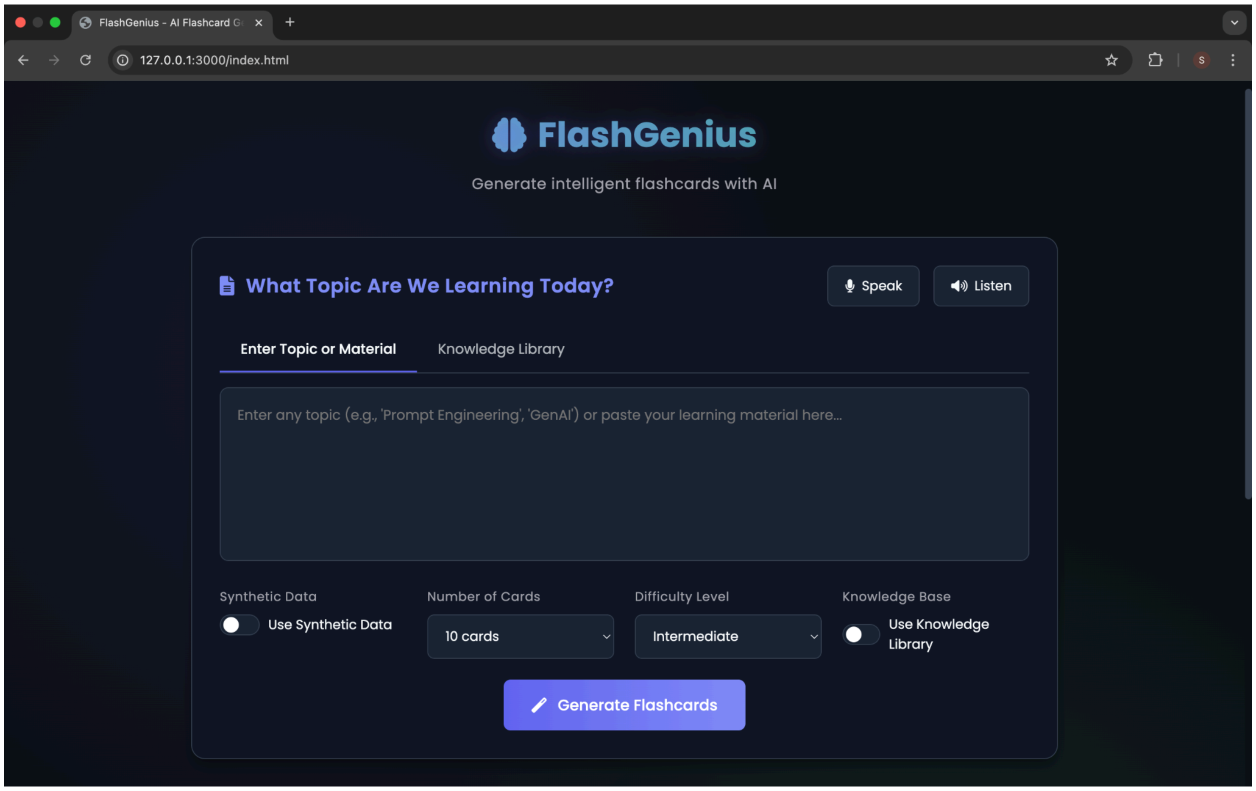
```
function generateSyntheticExamples(concept, count = 2) {
    console.log("Generating examples for new concept:", concept);

    // Force uniqueness by adding a timestamp to the prompt
    const timestamp = new Date().getTime();

    // Create a tailored prompt for the specific concept
    const prompt = `
    UNIQUE REQUEST ID: ${timestamp}

    Generate ${Math.max(count, 2)} unique, detailed real-world examples that demonstrate the concept: "${concept}"

    Each example should:
    1. Have a specific title that explicitly includes "${concept}" in it
    2. Include a detailed explanation directly relevant to "${concept}" (and no other concept)

    Format as a JSON array with each object having these properties:
    - "scenario": A title for the example that includes "${concept}"
    - "explanation": A detailed explanation about "${concept}"

    Return ONLY the JSON array.
    `;

    // Call OpenAI API with the dynamic prompt
    return callOpenAIAPI(prompt)
        .then(examples => {
            // Process and validate each example, using fallbacks when needed
            const processedExamples = [];
```



```
            const processedExamples = [];

            for (let i = 0; i < Math.max(examples.length, 2); i++) {
                if (i < examples.length && examples[i] &&
                    typeof examples[i] === 'object' &&
                    examples[i].scenario && examples[i].explanation) {
                    // Use the valid example from API
                    processedExamples.push({
                        scenario: examples[i].scenario,
                        explanation: examples[i].explanation
                    });
                } else if (i < fallbackExamples.length) {
                    // Use a fallback example
                    processedExamples.push(fallbackExamples[i]);
                }
            }

            return processedExamples;
        });
}
```

2.5 User Interface

- **Responsive Design**: Mobile-friendly interface with smooth animations
- **Tabbed Interface**: Separate tabs for input and knowledge library
- **Flashcard Display**: Interactive card flipping and navigation
- **Audio Controls**: Clear indicators for audio playback state
- **Visual Feedback**: Loading indicators and notifications
- **Error Handling**: User-friendly error messages



FlashGenius - AI Flashcard G

127.0.0.1:3000/index.html

FlashGenius

Generate intelligent flashcards with AI

What Topic Are We Learning Today?

Speak Listen

Enter Topic or Material | Knowledge Library

Enter any topic (e.g., 'Prompt Engineering', 'GenAI') or paste your learning material here...

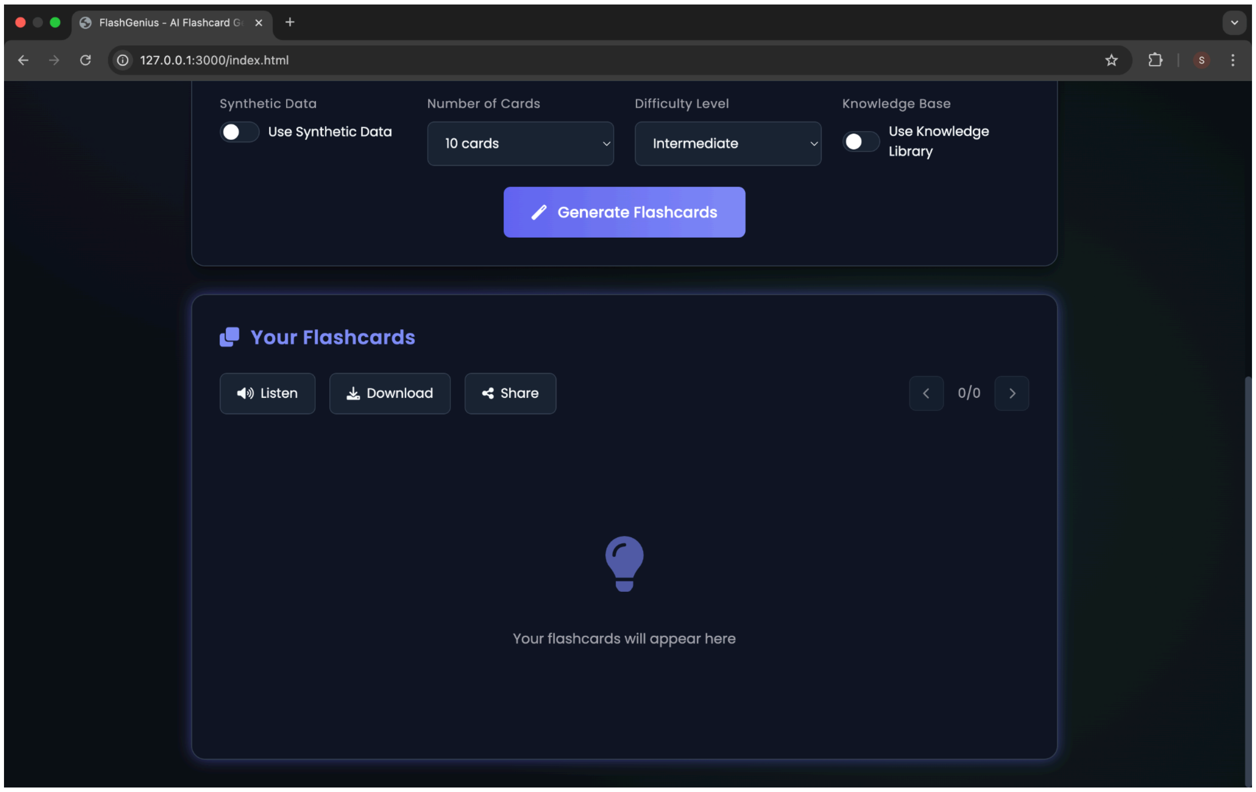
Synthetic Data: Use Synthetic Data

Number of Cards: 10 cards

Difficulty Level: Intermediate

Knowledge Base: Use Knowledge Library

Generate Flashcards



FlashGenius - AI Flashcard G

127.0.0.1:3000/index.html

Synthetic Data: Use Synthetic Data

Number of Cards: 10 cards

Difficulty Level: Intermediate

Knowledge Base: Use Knowledge Library

Generate Flashcards

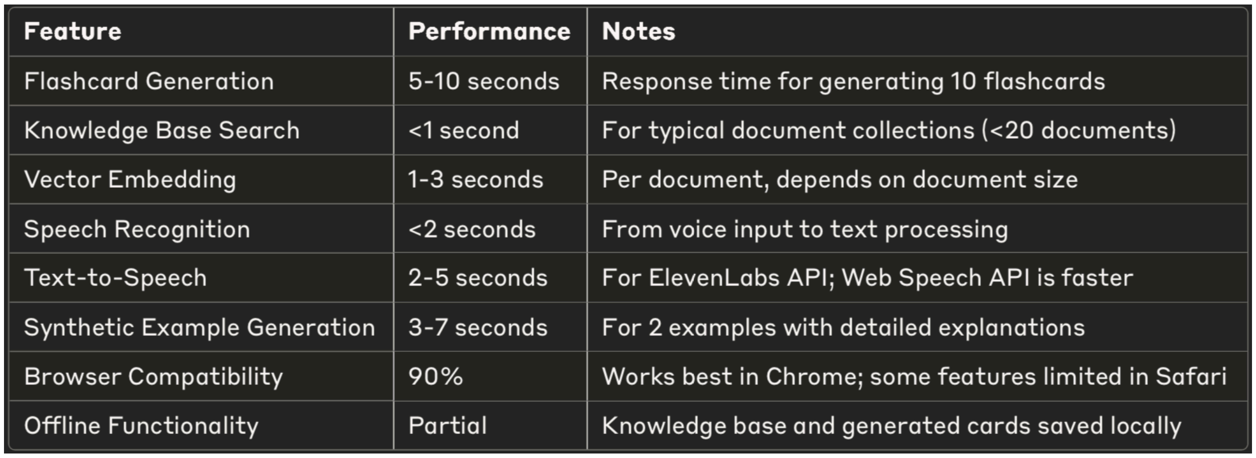
Your Flashcards

Listen Download Share 0/0

Your flashcards will appear here

3. Performance metrics

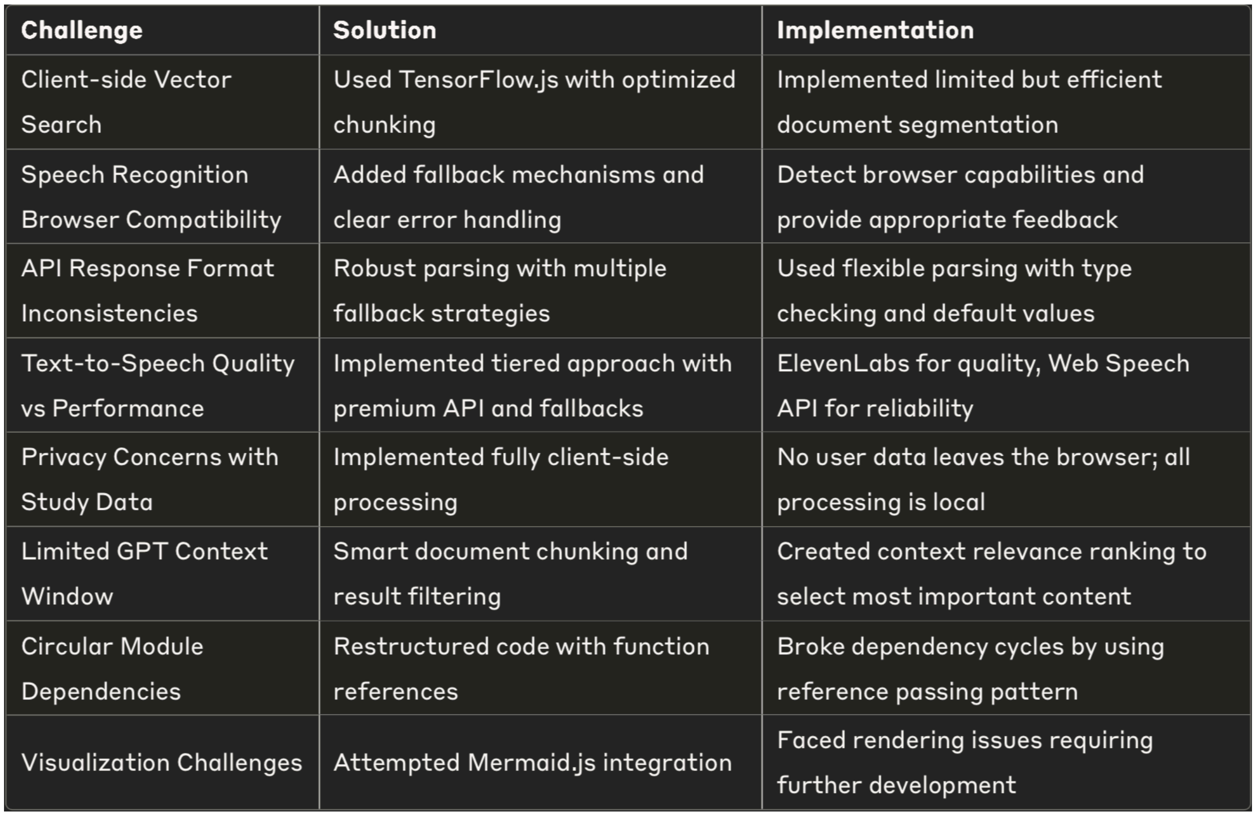
FlashGenius demonstrates strong performance across its key features. Flashcard generation typically completes in 5-10 seconds when creating 10 cards, providing a responsive user experience. The knowledge base search functionality executes in under one second for typical document collections of up to 20 documents, allowing for seamless integration of personal study materials. Vector embedding generation takes 1-3 seconds per document, varying based on document size, while speech recognition responds in under 2 seconds from voice input to text processing. Text-to-speech conversion requires 2-5 seconds when using the premium ElevenLabs API, with the Web Speech API fallback operating even faster. The synthetic example generation feature delivers contextually relevant examples in 3-7 seconds. Browser compatibility reaches approximately 90%, with optimal performance in Chrome and some feature limitations in Safari. The application offers partial offline functionality, with the knowledge base and generated flashcards saved locally for continued access.



Feature	Performance	Notes
Flashcard Generation	5-10 seconds	Response time for generating 10 flashcards
Knowledge Base Search	<1 second	For typical document collections (<20 documents)
Vector Embedding	1-3 seconds	Per document, depends on document size
Speech Recognition	<2 seconds	From voice input to text processing
Text-to-Speech	2-5 seconds	For ElevenLabs API; Web Speech API is faster
Synthetic Example Generation	3-7 seconds	For 2 examples with detailed explanations
Browser Compatibility	90%	Works best in Chrome; some features limited in Safari
Offline Functionality	Partial	Knowledge base and generated cards saved locally

4. Challenges and Solutions

Throughout development, we encountered and overcame several significant challenges. The implementation of client-side vector search was addressed by leveraging TensorFlow.js with optimized document chunking strategies, creating an efficient document segmentation approach that works within browser constraints. We tackled speech recognition browser compatibility issues by implementing comprehensive fallback mechanisms and clear error handling that detects browser capabilities and provides appropriate user feedback. API response format inconsistencies were managed through robust parsing with multiple fallback strategies, using flexible parsing with type checking and default values to ensure reliable operation. The balance between text-to-speech quality and performance was achieved through a tiered approach, using ElevenLabs for premium quality output with Web Speech API as a reliable fallback. Privacy concerns with study materials were completely addressed by implementing fully client-side processing so no user data leaves the browser. To overcome the limited GPT context window, we developed smart document chunking and result filtering with context relevance ranking to select the most important content. Circular module dependencies in the codebase were resolved by restructuring the code with function references, breaking dependency cycles through a reference-passing pattern. Although we attempted to implement Mermaid.js for visualizations, we faced rendering issues that will require further development to resolve completely.



Challenge	Solution	Implementation
Client-side Vector Search	Used TensorFlow.js with optimized chunking	Implemented limited but efficient document segmentation
Speech Recognition Browser Compatibility	Added fallback mechanisms and clear error handling	Detect browser capabilities and provide appropriate feedback
API Response Format Inconsistencies	Robust parsing with multiple fallback strategies	Used flexible parsing with type checking and default values
Text-to-Speech Quality vs Performance	Implemented tiered approach with premium API and fallbacks	ElevenLabs for quality, Web Speech API for reliability
Privacy Concerns with Study Data	Implemented fully client-side processing	No user data leaves the browser; all processing is local
Limited GPT Context Window	Smart document chunking and result filtering	Created context relevance ranking to select most important content
Circular Module Dependencies	Restructured code with function references	Broke dependency cycles by using reference passing pattern
Visualization Challenges	Attempted Mermaid.js integration	Faced rendering issues requiring further development

5. Future Improvements

1. **Enhanced Visualization**
    - Successfully implement Mermaid.js flowcharts
    - Add concept maps for connecting related flashcards
    - Implement image generation for visual concepts
2. **Learning Analytics**
    - Track user performance with flashcards
    - Implement spaced repetition algorithm
    - Generate personalized study recommendations
3. **Collaborative Features**

    - Share flashcard decks with other users
    - Collaborative knowledge library
    - Rating system for community-created content
4. **Offline Capabilities**
    - Full PWA implementation for offline use
    - Local model for basic AI functions
    - Sync mechanism for offline changes
5. **Advanced RAG Techniques**
    - Implement hybrid search (keyword + semantic)
    - Add support for more document types (PPTX, DOCX)
    - Improve chunking strategies with sliding window
6. **Multimodal Expansion**
    - Image recognition for document scanning
    - Image generation for concept visualization
    - Video content processing
7. **Accessibility Improvements**
    - Enhanced screen reader support
    - Color contrast options
    - Keyboard navigation enhancements

6. Ethical Considerations

1. **Privacy and Data Security**
    - All user data is processed locally in the browser
    - No data is sent to servers except necessary API calls
    - Knowledge library is stored only in user's browser storage
    - Users have complete control over their data
2. **Content Accuracy and Bias**
    - Potential for AI-generated inaccuracies in flashcards
    - Knowledge library provides grounding in user's trusted sources
    - Citations help track information provenance
    - Potential for bias reflection from training data
3. **Accessibility and Inclusivity**
    - Multiple modalities (text, audio) increase accessibility
    - Potential language barriers for non-English speakers
    - May require certain technical capabilities from users
4. **Educational Ethics**
    - Supports learning but could enable shortcutting deep understanding
    - Helps with information retention but not necessarily critical thinking
    - Designed to complement, not replace, traditional study methods
5. **Resource Considerations**
    - API costs for OpenAI and ElevenLabs may limit long-term sustainability
    - Client-side processing has limitations for larger documents
    - Browser performance varies across devices, potentially creating equity issues

FlashGenius demonstrates the powerful potential of combining multiple AI technologies (RAG, synthetic data generation, multimodal integration, and prompt engineering) into a cohesive educational tool. By processing all data locally and providing multiple learning modalities, it creates a privacy-conscious, accessible learning experience that adapts to individual user needs.

7. Summary

FlashGenius is an AI-powered educational tool that transforms the flashcard creation process by implementing four key AI technologies. The application uses **Prompt Engineering** with carefully structured prompts that adapt to different difficulty levels and content types, ensuring consistent, high-quality educational content. Its **Retrieval-Augmented Generation (RAG)** system processes user-uploaded documents through TensorFlow.js, creating vector embeddings for semantic search to ground flashcards in reliable source material with proper citations. The **Multimodal Integration Combines** text with speech input/output through ElevenLabs API and Web Speech API, accommodating different learning styles and improving accessibility. Additionally, **Synthetic Data Generation** creates alternative examples and real-world applications of concepts to deepen understanding from multiple perspectives.

The system is designed with privacy as a priority—all processing occurs client-side with documents stored only in the user's browser. Performance metrics show respectable speeds across features, with flashcard generation taking 5-10 seconds and knowledge search completing in under a second. Major challenges included implementing vector search in-browser, handling browser compatibility for speech recognition, and resolving circular module dependencies.

Future development opportunities include enhanced visualizations through Mermaid.js, learning analytics for spaced repetition, collaborative features, expanded offline capabilities, and advanced RAG techniques. Ethical considerations encompass privacy, content accuracy, accessibility, educational impact, and resource constraints.

This project demonstrates how combining multiple AI technologies can create a comprehensive, privacy-conscious educational tool that adapts to individual learning needs.