

Docky

GPT Code Snippets for Documentation

George Botros · Christian Ishimwe · Matthew Yin · Michael Imevbore
CIS 3500 Final Project

Problem & Solution

Problem

- Documentation sites often lack tailored, contextual code examples.
- Developers waste time hunting for sample code.
- Learners and cross-language devs struggle to bridge conceptual gaps.

Solution

- Docky Chrome extension
- On-page AI-powered snippet generator
- Users select context and language

Key Features

- **Context-Aware Generation**
 - Extracts the exact docs text you're ready
- **Language & Complexity Control**
 - Choose any language and difficulty
- **Formatting Options**
 - Indentation, blank lines, explanatory comments on demand
- **Non-Intrusive UI**
 - Popup UI when you use the extension that shows you the generated snippet

Docky

Selected Text:

```
createContext createContext lets you create a context that components can provide or read. const SomeContext = createContext(defaultValue) Reference createContext(defaultValue) SomeContext.Provider SomeContext.Consumer Usage Creating context Importing
```

OpenAI API Key:

..... Save Key

Select Language:

Python JavaScript C++ Java TypeScript

Select Complexity Level:

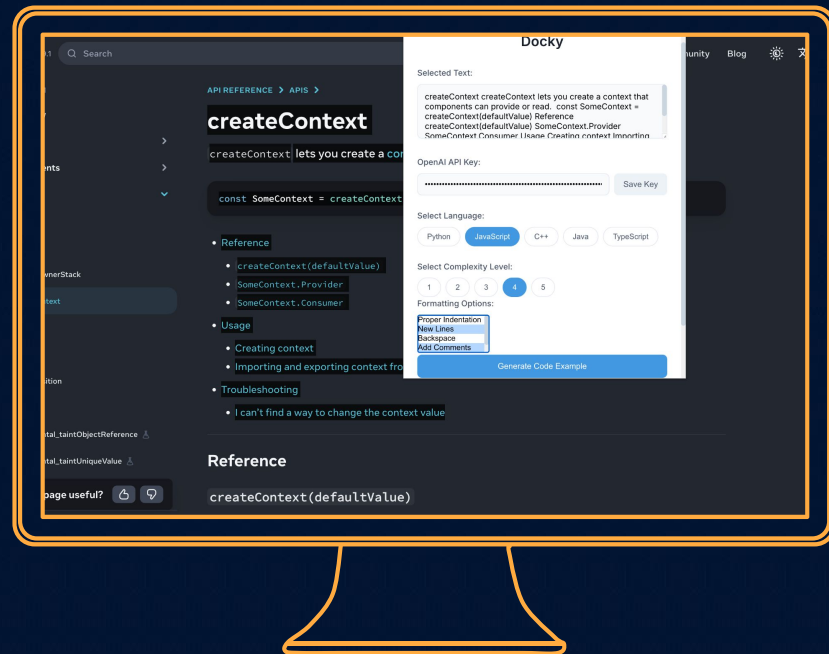
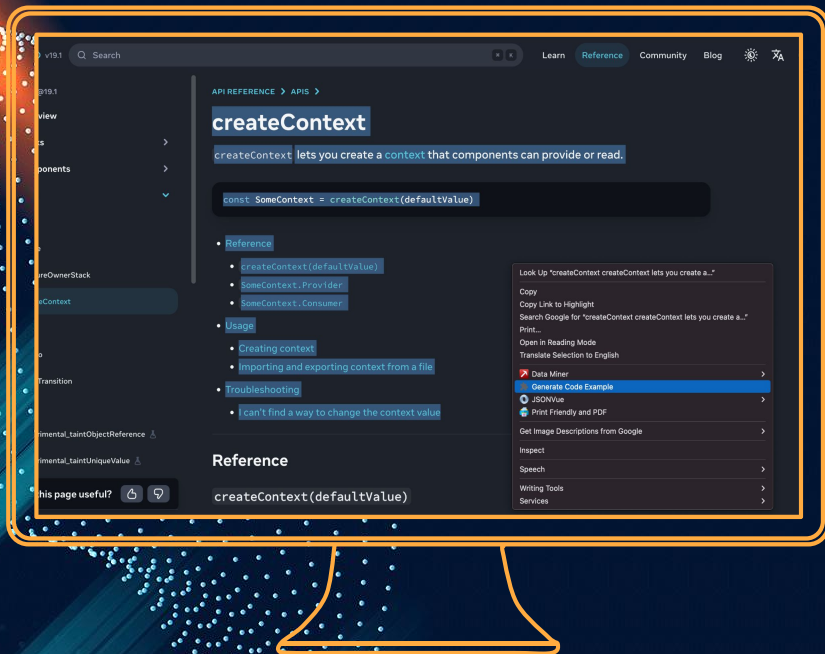
1 2 3 4 5

Formatting Options:

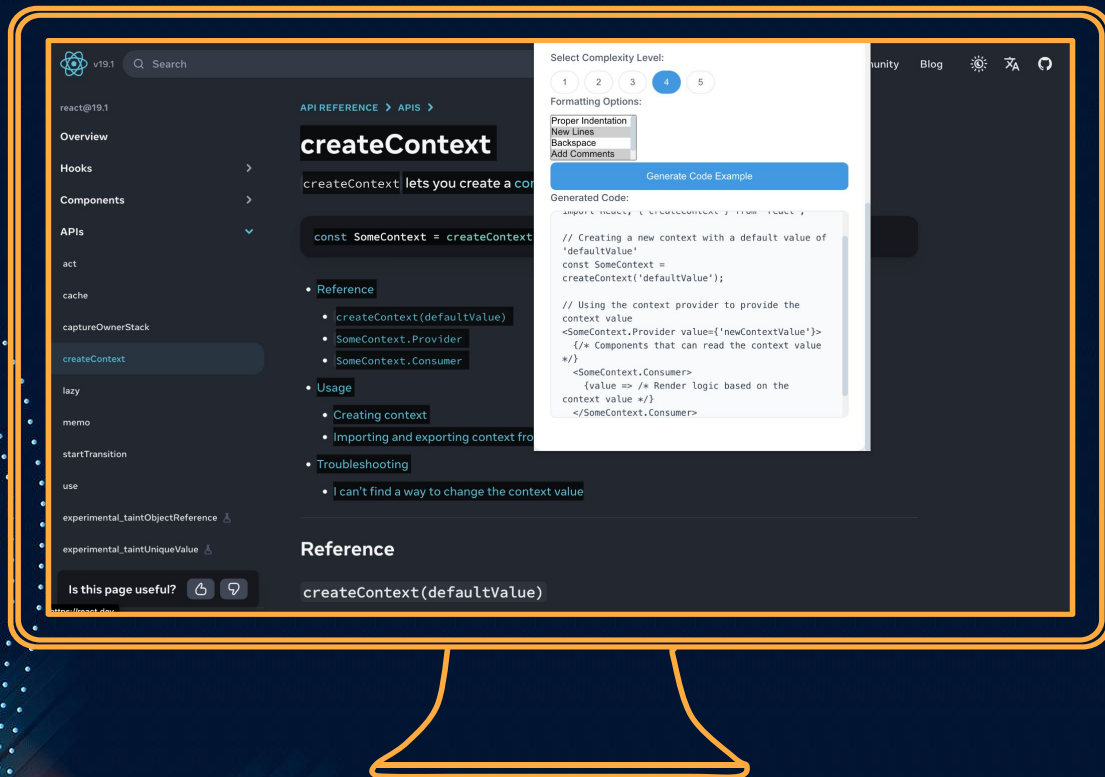
Proper Indentation
New Lines
Backspace
Add Comments

Generate Code Example

SCREENSHOTS



SCREENSHOTS



Architecture

- **Popup (UI Module)**
 - popup.html + popup.js
- **Reads/saves API key**
 - popup.js
- **AI Service (aiService.js)**
 - generateCodeSnippet() → wraps OpenAI Chat API call
 - cleanupCodeSnippet() → strips fences, trims
- **Background + Storage**
 - background.js for context menu
 - chrome.storage.local stores API key

```
***
const OPENAI_API_URL = 'https://api.openai.com/v1/chat/completions';

/**
 * Generates a code snippet based on the provided documentation context and user
 * preferences.
 *
 * @param {Object} options
 * @param {string} options.prompt - Your OpenAI API key. If not provided, the
 * widget on.
 * @param {string} [options.language='JavaScript'] - Desired programming language
 * for the snippet.
 * @param {string} [options.complexity='intermediate'] - Complexity level:
 * 'beginner', 'intermediate', 'advanced'.
 * @param {string} [options.model='gpt-3.5-turbo'] - OpenAI model to use.
 * @param {number} [options.temperature=0.2] - Sampling temperature.
 * @param {number} [options.maxTokens=500] - Max tokens in response.
 * @param {AbortSignal} [options.signal] - Optional signal to abort the request.
 * @returns {Promise<string>} The generated code snippet.
 */
export async function generateCodeSnippet({
  apiKey,
  contextText,
  language = 'JavaScript',
  complexity = 'intermediate',
  model = 'gpt-3.5-turbo',
  temperature = 0.2,
  maxTokens = 500,
  signal,
}) {
  if (!apiKey) {
    throw new Error('OpenAI API key is required!');
  }

  const systemMessage = {
    role: 'system',
    content: 'You are a helpful assistant that generates code snippets based on
    provided documentation context. ',
  };

  const userMessage = {
    role: 'user',
    content: contextText,
  };

  Please generate a ${complexity} level ${language} code example demonstrating the
  above context. Only return the code snippet.

  };

  const response = await fetch(OPENAI_API_URL, {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
      Authorization: `Bearer ${apiKey}`,
    },
    body: JSON.stringify({
      model,
      messages: [systemMessage, userMessage],
      temperature,
      max_tokens: maxTokens,
    }),
    signal,
  });

  if (!response.ok) {
    const errorText = await response.text();
    throw new Error('OpenAI API error: ' + response.status + ' ' + errorText);
  }

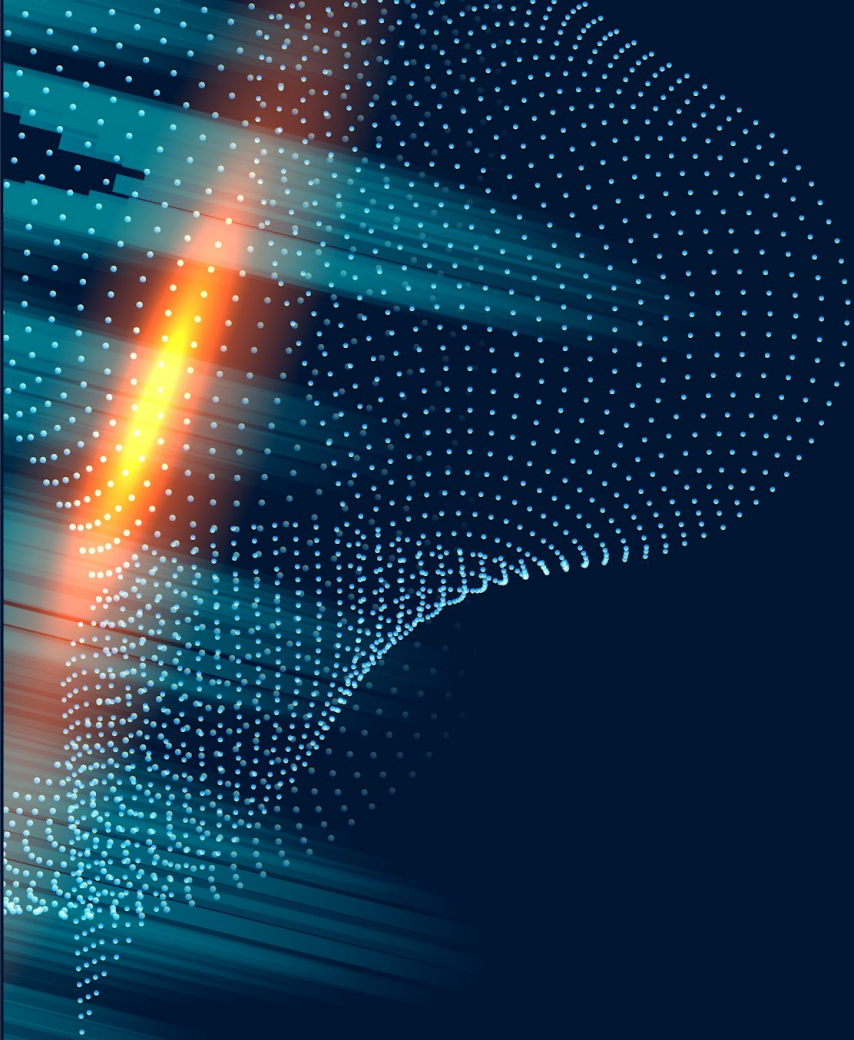
  const data = await response.json();
  const rawSnippet = data.choices?.[0]?.message?.content;
  if (!rawSnippet) {
    throw new Error('No content returned from OpenAI API!');
  }

  return rawSnippet.trim();
}

/**
 * Cleans up % raw code snippet by removing backticks if present.
 *
 * @param {string} raw - The raw snippet, possibly wrapped in triple backticks.
 * @returns {string} The cleaned code snippet.
 */
export function cleanupCodeSnippet(raw) {
  return raw
    .trim()
    .replace(/```/g, '')
    .replace(/`/g, '')
    .replace(/`/g, '')
    .replace(/`/g, '')
    .trim();
}
```

Challenges & Technical Decisions

- **Trouble with Module Loading in the Extension**
 - Chrome pop-up scripts default to classic `<script>`—we needed full ES-module support
 - Added `type="module"` to `popup.html` and verified correct relative paths for all imports
- **Robust Prompt Engineering**
 - Defined a clear system prompt for consistent coding style
 - Layered user prompts to include language, complexity, and optional formatting directives
 - Iterated on prompt wording to minimize “hallucinations” and maximize snippet relevance
- **Comprehensive Test Environment**
 - Autoloaded `.env` variables in Jest via a dedicated `jest.setup.js` so we can use API keys



Thanks!