

AI Integration for Web Development

I. Introduction to API Calling (Day 1)

Large Language Models (LLMs) are built upon massive computational systems due to their requirements to process large data each second. Because of this, companies usually host their own private servers where it runs all data processing and matrix calculations. An API acts as a key for developers to access just the model to work with, without risking any tamperings of the main servers.

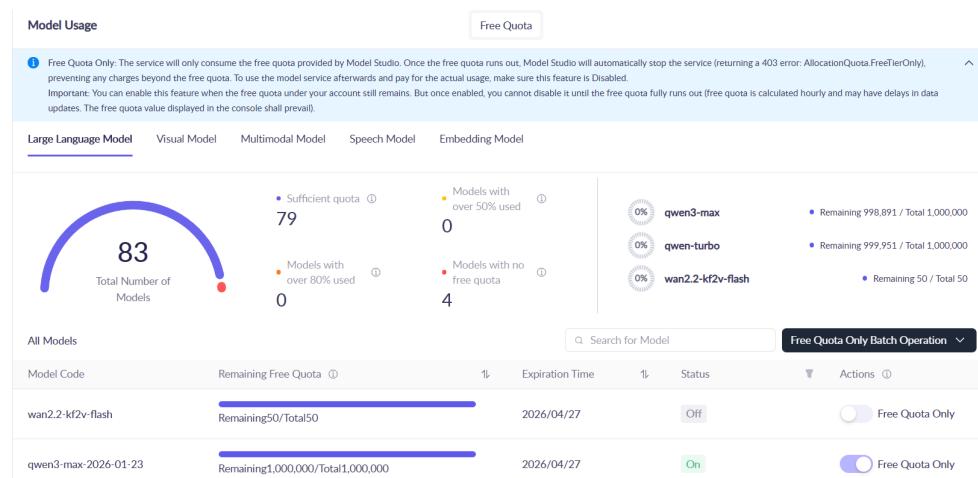
A. API Functions Towards Interface Standards and Security

An API sets the standard for all users on how they interact with the LLM. Developers using the API key to access the model obtain a standardized collection of functions and tools tailored to the use of that specific model. We can think of this as adding an extra library to include in an IDE, which lets developers write predictable code.

Having access only to the model also isolates your data and code from other developers and vice versa. Additionally, any improvements made on the AI model behind the API would not have any affect on a developer's code.

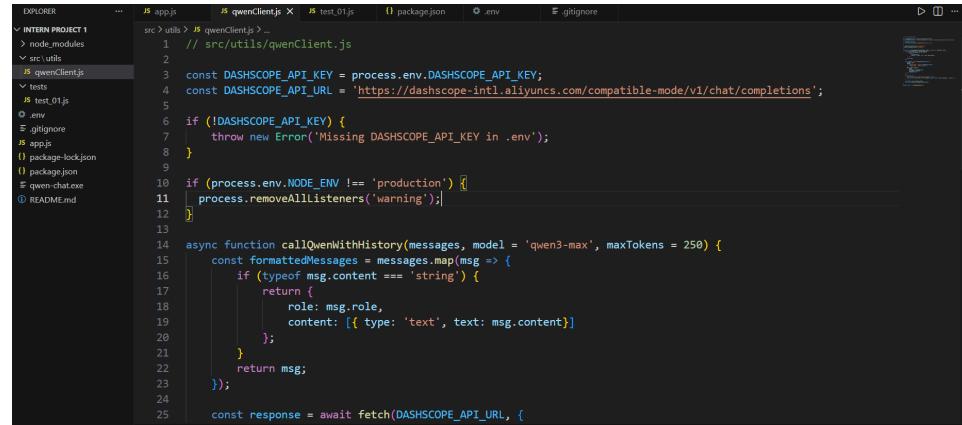
B. Basics of API Key Application

In order to carry out a test of working with an existing LLM, I chose the company Alibaba which has an abundance of working AI models comparable to OpenAI's chatGPT. After creating a free account and obtaining an API key, we gain access to more than 70 AI models with a **free** quota of 1 million output tokens.



Before working with any of those models, an essential step is to store the private API key in the environment variables or a .env file. This step ensures security on the data from other developers and avoids getting banned from the server for any abuse.

After creating a project folder, we use java script to create a default qwenClient.js file to call Qwen. This enables all other applications to access the model without code repetition. It should handle authentication, message formatting specific to Qwen, error handling, etc. Standardizing the input format is important since most LLMs use a multimodal input format.



```

EXPLORER          JS app.js      JS qwenClient.js     JS test_01.js    package.json   .env       .gitignore
> INTERN PROJECT 1
> node_modules
> src
  > utils
    > qwenClient.js
  > tests
    > test_01.js
  > .env
  > .gitignore
  > app.js
  > package-lock.json
  > package.json
  > qwen-chat.exe
  > README.md

src > utils > qwenClient.js > ...
1 // src/utils/qwenClient.js
2
3 const DASHSCOPE_API_KEY = process.env.DASHSCOPE_API_KEY;
4 const DASHSCOPE_API_URL = 'https://dashscope-intl.aliyuncs.com/compatible-mode/v1/chat/completions';
5
6 if (!DASHSCOPE_API_KEY) {
7   throw new Error('Missing DASHSCOPE_API_KEY in .env');
8 }
9
10 if (process.env.NODE_ENV !== 'production') {
11   process.removeAllListeners('warning');
12 }
13
14 async function callQwenWithHistory(messages, model = 'qwen3-max', maxTokens = 250) {
15   const formattedMessages = messages.map(msg => {
16     if (typeof msg.content === 'string') {
17       return {
18         role: msg.role,
19         content: [{ type: 'text', text: msg.content }]
20       };
21     }
22     return msg;
23   });
24
25   const response = await fetch(DASHSCOPE_API_URL, {
26     method: 'POST',
27     headers: {
28       'Content-Type': 'application/json'
29     },
30     body: JSON.stringify({
31       model,
32       messages: formattedMessages,
33       max_tokens: maxTokens
34     })
35   });
36
37   const data = await response.json();
38   return data.choices[0].text;
39 }
40
41 module.exports = callQwenWithHistory;
42
43 // This is a template file. You can edit it and start building your project.
44 // To run this file you can use the command "node app.js" or "node index.js"
45 // depending on your file structure.
46
47 // If you have any questions about this template please refer to https://bit.ly/3JNnDZC
48
49 // Happy coding!
50
51
52
53
54
55
56

```

[qwenClient.js](#) exports a function ‘callQwenWithHistory’ which works to analyze a string of inputs using the default ‘qwen3-max’ model. Calling the function from any external .js file like the example below can be used to create a command line interface (CLI) where Qwen is integrated in. In this case, we can hold a conversation with the AI model with an expected output under 500 tokens.



```

JS app.js      JS qwenClient.js     JS test_01.js    package.json   .env       .gitignore
app.js > ...
16
17 // Creating a readline interface
18 const rl = readline.createInterface({
19   prompt: 'You: ',
20   input: process.stdin,
21   output: process.stdout
22 });
23
24 console.log('Qwen3-Max Chat Console (press Ctrl + C to exit)!\\n');
25
26 rl.prompt();
27
28 let messages = [];
29
30 rl.on('line', async (input) => {
31   if (input.trim() === '') {
32     rl.prompt();
33     return;
34   }
35
36   messages.push({ role: 'user', content: input});
37
38 try {
39   console.log('Qwen is thinking...\\n');
40
41   const response = await callQwenWithHistory(messages);
42
43   messages.push({ role: 'assistant', content: response });
44
45   console.log(`Qwen3-Max: ${response}\\n`);
46 } catch(err) {
47   console.error(`Error: ${err.message}\\n`);
48 }
49
50 rl.prompt();
51 });
52
53 rl.on('close', () => {
54   console.log(`\\nShut down.\\n`);
55   process.exit(0);
56 })

```

```
Qwen3-Max Chat Console (press Ctrl + C to exit)!  
You: what is 5+5  
Qwen is thinking...  
Qwen3-Max: 5 + 5 equals 10.  
You: |
```

II. AI Integration In Inputting Product Attributes (Day 2)

A. Problem Scenario