Summary Plugin with OpenAI API

Index

Introduction:

In the rapidly evolving landscape of web content creation, the need for efficient tools to enhance productivity and streamline workflows is paramount. This project addresses this need by seamlessly integrating a summary generation plugin with Open AI API into the WordPress editor.

Project Rationale:

- Content Abundance: As content creators, bloggers, and website administrators grapple with an abundance of information, distilling key points becomes essential for audience engagement.

- Automation and AI: Leveraging the power of OpenAI's language models allows for the automated generation of concise and coherent summaries, providing a valuable resource for content creators seeking to optimize their workflow.

- UserFriendly Integration: The project's primary goal is to provide a userfriendly and intuitive solution. By embedding the summary generation functionality directly into the WordPress editor, users can effortlessly obtain summaries for their posts with a single click.

Project Requirements:

Before initiating the project, ensure you have the following prerequisites and requirements in place:

- OpenAI API Key:
  Acquire a valid OpenAI API key. This key is essential for authenticating and authorizing requests to the OpenAI language models. If you don't have an API key, you can obtain one from the OpenAI platform.

- OpenAI API Endpoint:
  The OpenAI API endpoint is the URL to which requests are made. For the GPT3.5 Turbo model, use the following endpoint:

  https://api.openai.com/v1/engines/gpt3.5turbo/completions

  Ensure your API key is appropriately configured to access this endpoint.

- WordPress Installation:
  The project assumes a WordPress installation. Make sure you have a working WordPress site with administrative access.

- PHP Version:
  The PHP version on your server should be 5.6 or higher. The project utilizes PHP to handle serverside functionality.

Architecture Overview:

The project architecture consists of two fundamental components: a serverside WordPress plugin (PHP) and a clientside script(JavaScript).

1. ServerSide Plugin (PHP):
   AJAX Endpoint: Creates an AJAX endpoint for handling summary requests securely.
   OpenAI API Integration: Utilizes the OpenAI API to generate summaries based on the post content.

Script Enqueuing: Ensures the proper inclusion of the clientside JavaScript script.

2. ClientSide Script (JavaScript):
Button Click Handling: Listens for button clicks and triggers the summary generation process.
AJAX Request: Sends asynchronous requests to the serverside endpoint, initiating the summary generation.
User Interface Update: Dynamically updates the post editor with the generated summary, ensuring a smooth user experience.

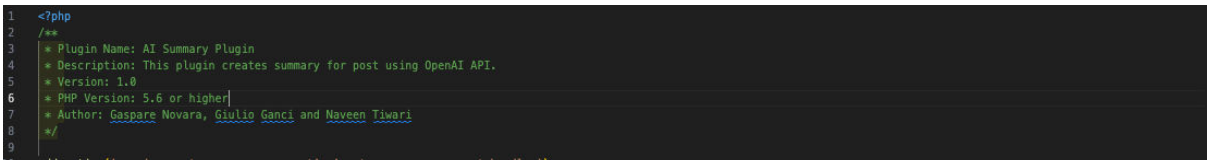
Code Explanation:

The project is divided in two files, the PHP file and the JavaScript file.

PHP File:

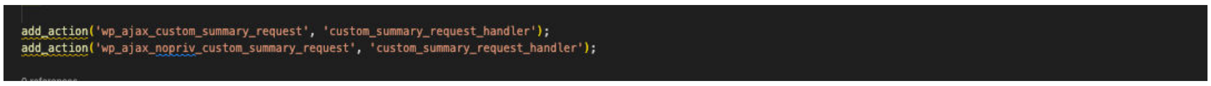
The PHP file ("summary_plugin.php") contains functions to create the serverside endpoint, handle OpenAI API requests, enqueue scripts, and add the custom button to the post editor. Below is the brief description.

1. Plugin Header: This section provides information about the plugin, including its name, description, version, and author.



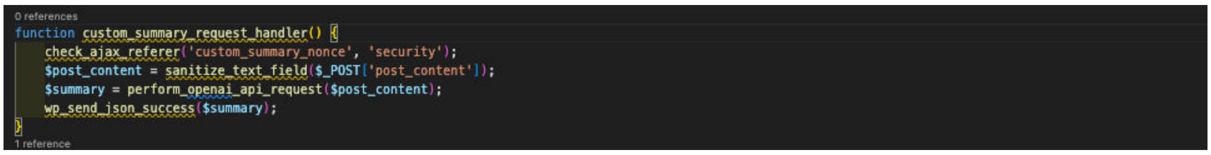
```php
<?php
/**
 * Plugin Name: AI Summary Plugin
 * Description: This plugin creates summary for post using OpenAI API.
 * Version: 1.0
 * PHP Version: 5.6 or higher
 * Author: Gaspare Novara, Giulio Ganci and Naveen Tiwari
 */
```

2. AJAX Endpoint: Two actions ("wp_ajax_custom_summary_request" and "wp_ajax_nopriv_custom_summary_request") are hooked to the "custom_summary_request_handler" function. This function handles AJAX requests for the custom summary.



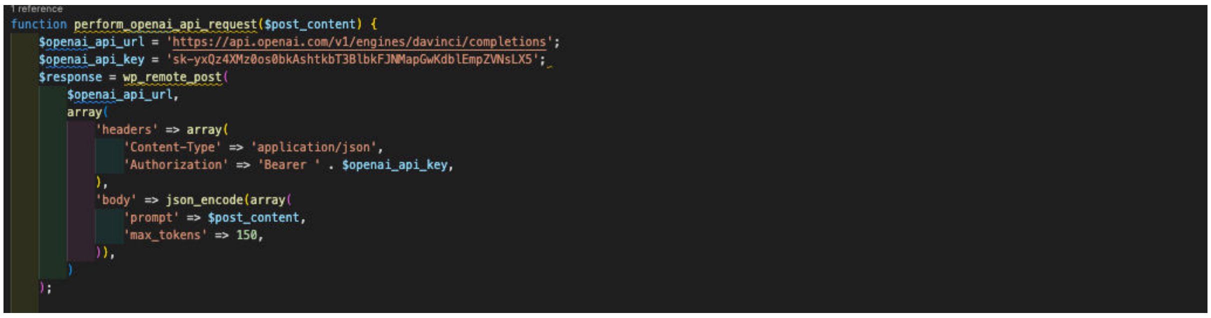
```php
add_action('wp_ajax_custom_summary_request', 'custom_summary_request_handler');
add_action('wp_ajax_nopriv_custom_summary_request', 'custom_summary_request_handler');
```

3. Custom Summary Request Handler: This function checks the AJAX nonce for security, sanitizes the post content from the AJAX request,performs the Open AI API request using "perform_openai_api_request" function, and sends the summary as a JSON response.



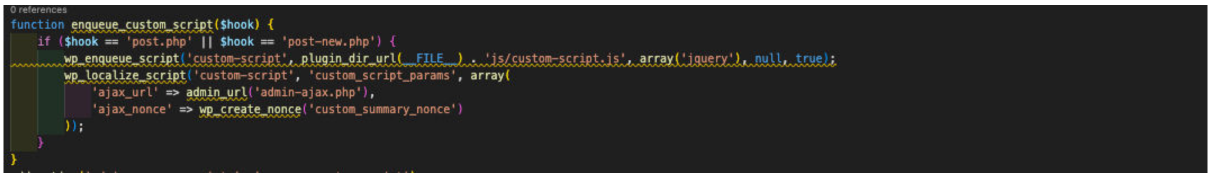
```
0 references
function custom_summary_request_handler() {
    check_ajax_referer('custom_summary_nonce', 'security');
    $post_content = sanitize_text_field($_POST['post_content']);
    $summary = perform_openai_api_request($post_content);
    wp_send_json_success($summary);
}
1 reference
```

4. Perform OpenAI API Request: This function constructs an HTTP POST request to the OpenAI API, including the prompt (post content) and maximum tokens for the summary. It handles errors and decodes the API response.



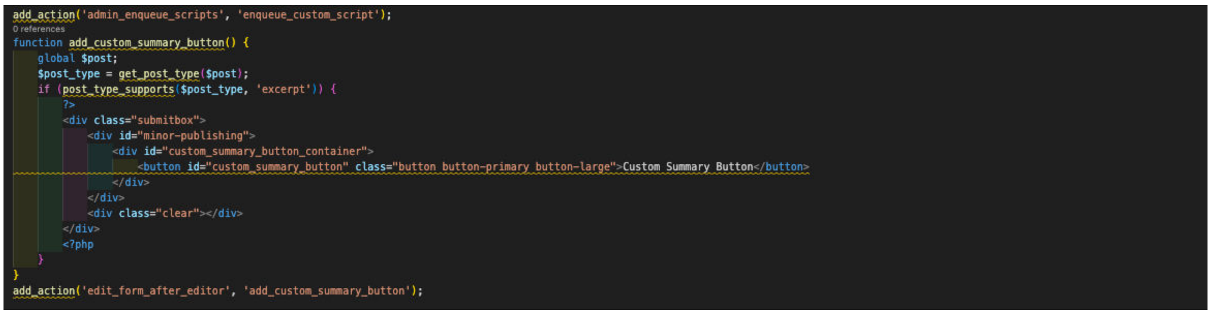
```
1 reference
function perform_openai_api_request($post_content) {
    $openai_api_url = 'https://api.openai.com/v1/engines/davinci/completions';
    $openai_api_key = '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';
    $response = wp_remote_post(
        $openai_api_url,
        array(
            'headers' => array(
                'Content-Type' => 'application/json',
                'Authorization' => 'Bearer ' . $openai_api_key,
            ),
            'body' => json_encode(array(
                'prompt' => $post_content,
                'max_tokens' => 150,
            )),
        )
    );
```

5. Enqueue Script in WordPress: The "enqueue_custom_script" function is hooked to "admin_enqueue_scripts". It enqueues the custom JavaScript file ("customscript.js") with jQuery dependency and localizes the script with parameters like AJAX URL and nonce.



```
0 references
function enqueue_custom_script($hook) {
    if ($hook == 'post.php' || $hook == 'post-new.php') {
        wp_enqueue_script('custom-script', plugin_dir_url(__FILE__) . 'js/custom-script.js', array('jquery'), null, true);
        wp_localize_script('custom-script', 'custom_script_params', array(
            'ajax_url' => admin_url('admin-ajax.php'),
            'ajax_nonce' => wp_create_nonce('custom_summary_nonce')
        ));
    }
}
```

6. Add Custom Summary Button: The "add_custom_summary_button" function is hooked to "edit_form_after_editor". It adds the custom summary button below the post description if the post type supports excerpts.



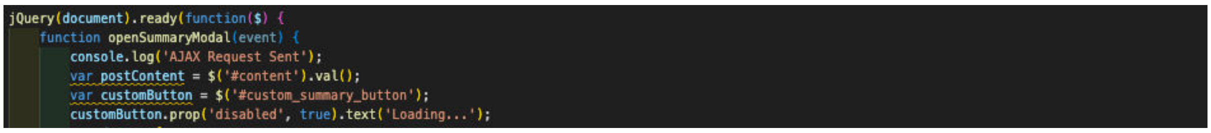
```php
add_action('admin_enqueue_scripts', 'enqueue_custom_script');
0 references
function add_custom_summary_button() {
    global $post;
    $post_type = get_post_type($post);
    if (post_type_supports($post_type, 'excerpt')) {
        ?>
        <div class="submitbox">
            <div id="minor-publishing">
                <div id="custom_summary_button_container">
                    <button id="custom_summary_button" class="button button-primary button-large">Custom Summary Button</button>
                </div>
            </div>
            <div class="clear"></div>
        </div>
        <?php
    }
}
add_action('edit_form_after_editor', 'add_custom_summary_button');
```

Now, we have a plugin that integrates a custom summary button with the OpenAI API, and the JavaScript file is enqueued to handle AJAX requests. The button appears below the post description, and when clicked, it triggers an AJAX request to fetch and display the summary.

JavaScript File:

The JavaScript file ("customscript.js") defines functions for handling button clicks, sending AJAX requests, and updating the post editor with the generated summary.

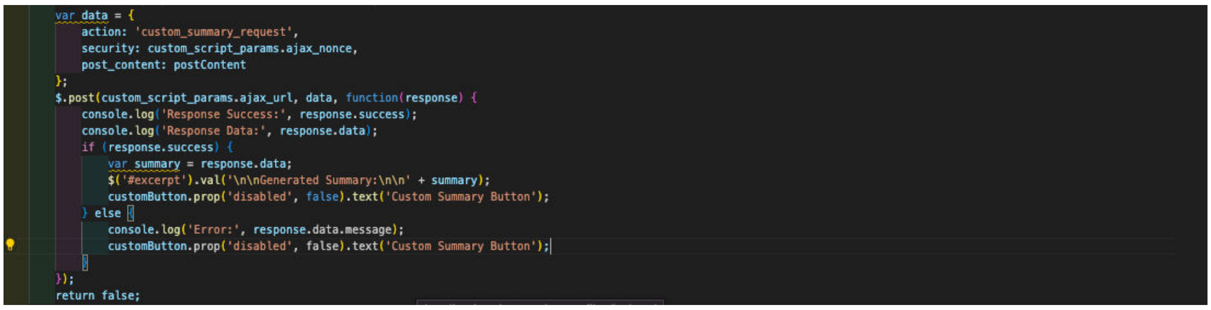
1. Document Ready Function: The code is wrapped in "jQuery(document).ready(function($) { ... });" to ensure that it runs after the document has been fully loaded.



```javascript
jQuery(document).ready(function($) {
    function openSummaryModal(event) {
        console.log('AJAX Request Sent');
        var postContent = $('#content').val();
        var customButton = $('#custom_summary_button');
        customButton.prop('disabled', true).text('Loading...');
```

2. openSummaryModal Function: This function is triggered when the custom summary button is clicked. It performs the following actions:

7. Logs an AJAX request confirmation to the console.
8. Retrieves the post content from the editor.
9. Disables the custom button and changes its text to "Loading...".
10. Constructs an AJAX data object with the action, security nonce, and post content.



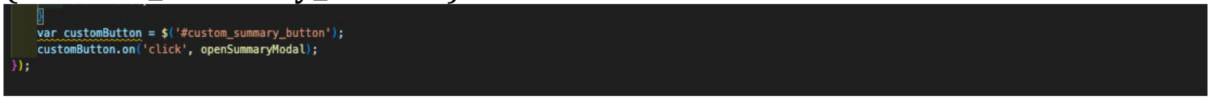
```javascript
        var data = {
            action: 'custom_summary_request',
            security: custom_script_params.ajax_nonce,
            post_content: postContent
        };
        $.post(custom_script_params.ajax_url, data, function(response) {
            console.log('Response Success:', response.success);
            console.log('Response Data:', response.data);
            if (response.success) {
                var summary = response.data;
                $('#excerpt').val('\n\nGenerated Summary:\n\n' + summary);
                customButton.prop('disabled', false).text('Custom Summary Button');
            } else {
                console.log('Error:', response.data.message);
                customButton.prop('disabled', false).text('Custom Summary Button');
            }
        });
        return false;
```

Makes a POST request to the WordPress serverside endpoint with the provided data.

Handles the serverside response:

- If successful, it updates the post editor's excerpt field with the generated summary and reenables the custom button.
- If there's an error, it logs the error message, reenables the custom button, and restores its original text.

3. Accessing Custom Button: Retrieves the custom button by its ID (“custom_summary_button”).



```
    }
    var customButton = $('#custom_summary_button');
    customButton.on('click', openSummaryModal);
});
```

4. Click Event Listener: Adds a click event listener to the custom button, triggering the “openSummaryModal” function when clicked.

This JavaScript code handles the AJAX request, updates the post editor's excerpt field with the generated summary, and provides feedback to the user.

Conclusion:

This project seamlessly integrates with WordPress, providing users with a convenient tool to generate summaries for their posts using the OpenAI API. The serverside PHP plugin handles AJAX requests, interacts with the OpenAI API, and enqueues the clientside JavaScript script. The JavaScript script manages button clicks, sends requests, and updates the post editor with the generated summary.