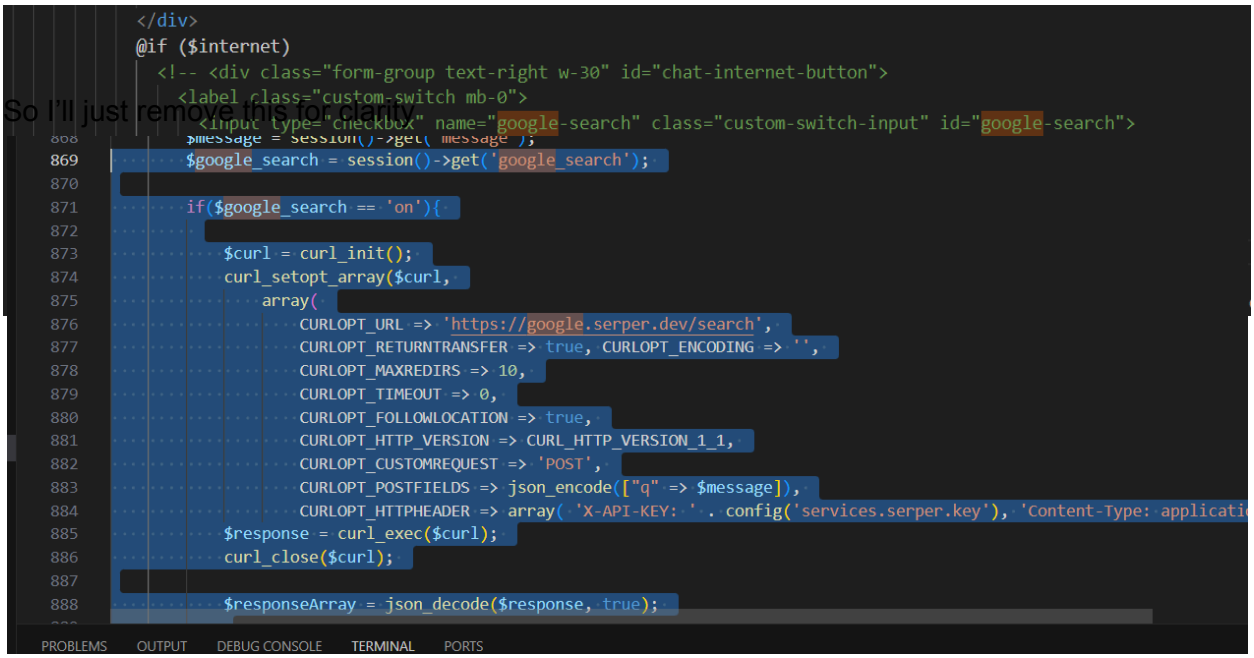
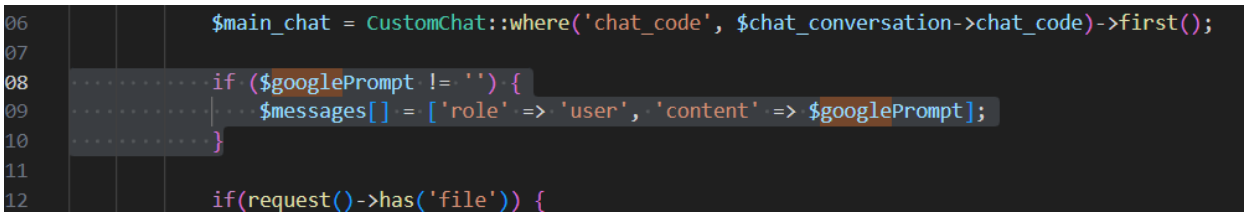
What should be remove:

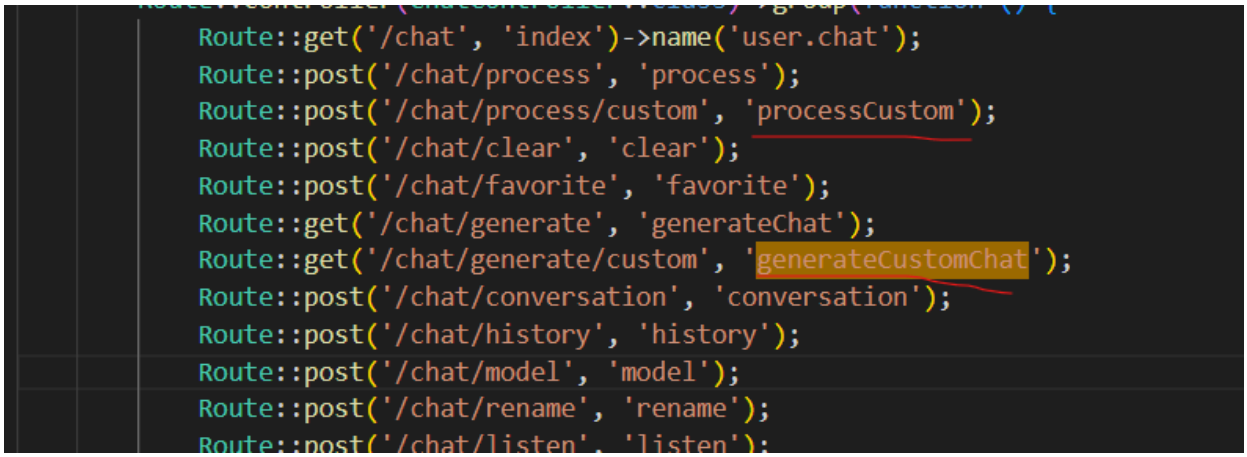
I notice this google-search is commented on so I guess we also don't need the api part of this?.





Also the error comes from this cause I believe it's not properly configured although we already have our own version of web access with web scraping feature so it makes sense we remove the default script google_search codes.

Also Just wanted to clarify this are test functions right? Maybe we should also remove this



```
Route::get('/chat', 'index')->name('user.chat');
Route::post('/chat/process', 'process');
Route::post('/chat/process/custom', 'processCustom');
Route::post('/chat/clear', 'clear');
Route::post('/chat/favorite', 'favorite');
Route::get('/chat/generate', 'generateChat');
Route::get('/chat/generate/custom', 'generateCustomChat');
Route::post('/chat/conversation', 'conversation');
Route::post('/chat/history', 'history');
Route::post('/chat/model', 'model');
Route::post('/chat/rename', 'rename');
Route::post('/chat/listen', 'listen');
```

What are the Needed on Web enable chat feature:

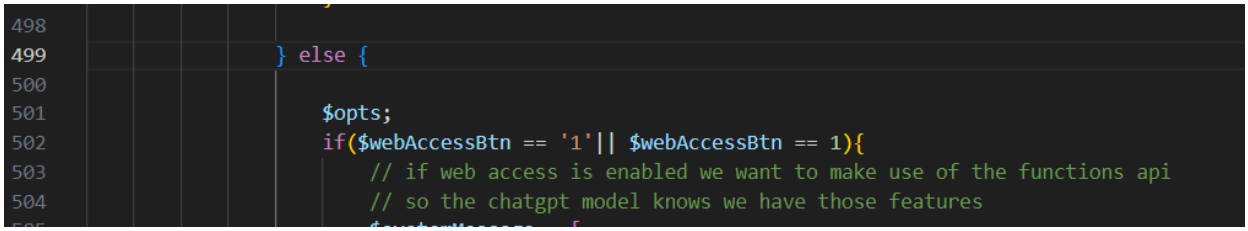
We should pass the web_acess at the request payload in Front end not on session, as it simplifies thing and reduce bug when two user with the same account log in’s

```
935  // Check textarea input
936  $(function () {
937    main_form.addEventListener("submit", event => {
938      event.preventDefault();
939      var webAccessBtn = $("#web_access_button").prop('checked') ? 1 : 0;
940      const message = input_text.value;
941      if (!message) {
942        toastr.warning('{{ __('Type your message first before sending') }}');
943        return;
944      }
945
946      appendMessage(user_avatar, "right", message, '', uploaded_image);
947      input_text.value = "";
948      process(message,webAccessBtn)
949    });
950
951  });
```

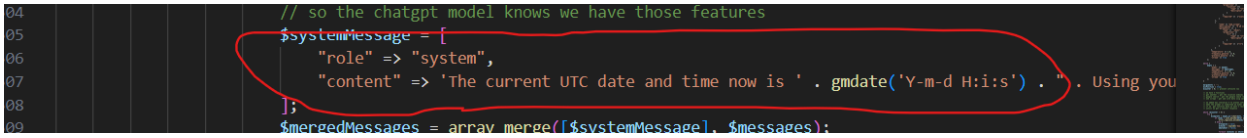
```
        // since the session is not always the same depending on user
cookies setting
        // we opt in to pass the web_access variable on each request
        $webAccessBtn = $request->web_access;
```

Then we get that variable on the laravel backend like this

This is the start of the web access, we check if it's enabled



We want to give the LLM(chatgpt model) to know what's the current time for reference so we pass it like this.



Then we setup the opts to be pass on the OpenAi class with the function api's but before that we should have variable that will hold the info about the chatgpt generation process like(if it needs to call a function to answer or it is ready to give output already)

```
                                    "name" => "get_current_weather",
                                    "description" =>  "Get the current
weather in a given location.",
                                    "parameters" =>  [
                                        "type" =>  "object",
                                        "properties" =>  [
                                            "location" => [
                                                "type" =>  "string",
                                                "description" => "The
location need weather info"
                                            ]
                                        ],
                                        "required" =>  ["location"]
                                    ]
                                ],
```

For simplicity you can just copy paste this opts with its prompt already created

```
                        $opts = [
                            'model' => $model,
                            'messages' => $mergedMessages,
                            'functions' => [
                                [
                                    "name" => "web_search",
                                    "description" =>  "A search engine.
useful for when you need to search the web. Please call the scrape
function when searching for news.",
                                    "parameters" =>  [
                                        "type" =>  "object",
                                        "properties" =>  [
                                            "query" => [
                                                "type" =>  "string",
                                                "description" => "The
information needed to search"
                                            ]
                                        ],
                                        "required" =>  ["query"]
                                    ]
                                ],
                                [
                                    "name" => "get_current_weather",
                                    "description" =>  "Get the current
weather in a given location.",
                                    "parameters" =>  [
                                        "type" =>  "object",
                                        "properties" =>  [
                                            "location" => [
                                                "type" =>  "string",
                                                "description" => "The
location need weather info"
                                            ]
                                        ],
                                        "required" =>  ["location"]
                                    ]
                                ],
```

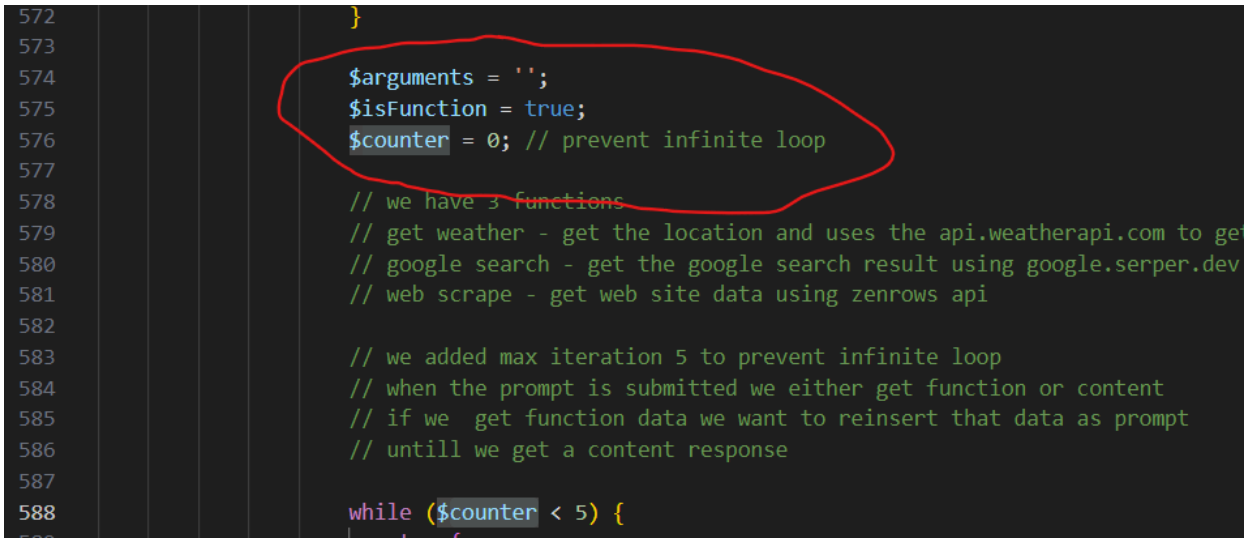
```
                                [
                                    "name" => "web_scraper",
                                    "description" =>  "A web scraper.
useful for when you need to scrape websites for additional information.
Most useful for when gathering information for news.",
                                    "parameters" =>  [
                                        "type" =>  "object",
                                        "properties" =>  [
                                            "url" => [
                                                "type" =>  "string",
                                                "description" => "The web
site url"
                                            ]
                                        ],
                                        "required" =>  ["url"]
                                    ]
                                ]
                            ],
                            'temperature' => 1.0,
                            'frequency_penalty' => 0,
                            'presence_penalty' => 0,
                            'stream' => true
                        ];

                    }else {
                        $opts = [
                            'model' => $model,
                            'messages' => $messages,
                            'temperature' => 1.0,
                            'frequency_penalty' => 0,
                            'presence_penalty' => 0,
                            'stream' => true
                        ];
                    }
```

$arguments holds the data extracted from the user input, sample
User: What is the weather like in New York city?
The arguments value would be “new your city” and this will be use by the weather function prompt that we give

$isFunction this is just a variable that will contain if the output of the first iteration is a function or a content. Function means it extract output from one of the following function we have either get weather, search google, or web scrape a site.

$counter there could be instance where the user question cannot be answered even using the functions couple of times. We need to limit it so for this we give it 5 max iteration before it gives up to prevent infinite loop.



We pass it to the openai class like this

Have this as reference on function api https://platform.openai.com/docs/guides/function-calling

This code should be self explanatory from this docs https://platform.openai.com/docs/guides/function-calling.
Basically we check if we get an output then check if it's a function or a content if it's a function we iterate again till we get the content.
Also we send a info to front end so user know what's the current iteration is ongoing (Getting weather for $variable, searching for $variable etc)

```
while ($counter < 5) {
    try {
```

```
                            $complete = $open_ai->chat($opts, function
($curl_info, $data) use (&$text, &$arguments, &$opts, &$counter,
&$isFunction, &$messages) {
                                    if ($obj = json_decode($data) and
$obj->error->message != "") {

error_log(json_encode($obj->error->message));
                                    } else {
                                        $response = null;
                                        $chunks = explode('data: ', $data);
                                        $dataSent = false;

                                        foreach ($chunks as $chunk) {
                                            $trimmed_chunk = trim($chunk);

                                            if (!empty($trimmed_chunk)) {
                                                $response =
json_decode($trimmed_chunk, true);
                                                $delta = null;
                                                if
(isset($response["choices"])) {
                                                    $delta =
$response["choices"][0]["delta"];
                                                }
                                                if
(isset($delta['function_call'])) {
                                                    if
(isset($delta['function_call']['arguments'])) {
                                                        $arguments .=
$delta['function_call']['arguments'];
                                                    }
                                                } else if
(isset($response["choices"][0]['finish_reason']) &&
$response["choices"][0]['finish_reason'] == 'function_call') {
                                                    $jsonData =
json_decode($arguments);
                                                    $process = '';
                                                    if
(isset($jsonData->location)) {
```

```
                                                $process = "Getting
weather location for $jsonData->location";
                                                echo 'data:
{"choices":[{"index":0,"delta":{"process":true,"content":"' . $process .
'"}}]}' . "\n\n";
                                                echo PHP_EOL;
                                                ob_flush();
                                                flush();
                                                array_push($messages,
['role' => 'assistant', 'content' => $process]);

                                                $content =
$this->getWeather($jsonData);
                                            } else if
(isset($jsonData->query)) {
                                                $process = "Searching
for $jsonData->query";
                                                echo 'data:
{"choices":[{"index":0,"delta":{"process":true,"content":"' . $process .
'"}}]}' . "\n\n";
                                                echo PHP_EOL;
                                                ob_flush();
                                                flush();
                                                array_push($messages,
['role' => 'assistant', 'content' => $process]);
                                                //

array_push($opts['messages'], ['role' => 'user', 'content' => 'when
searching if and only if you need more information to provide a more
complete answer Please scrape into the urls as additional research. Scrape
a url only once. If encounter problem scraping url, try other urls from
the web search.']);
                                                $content =
$this->webSearch($jsonData);
                                            } else if
(isset($jsonData->url)) {
                                                $process = "Reading
contents of $jsonData->url";
```

```
                                                    echo 'data:
{"choices":[{"index":0,"delta":{"process":true,"content":"' . $process .
'"}}]}' . "\n\n";
                                                    echo PHP_EOL;
                                                    ob_flush();
                                                    flush();
                                                    array_push($messages,
['role' => 'assistant', 'content' => $process]);

                                                    $content =
$this->webScrape($jsonData);
                                                }

                                                $arguments = '';
                                                ++$counter;


array_push($opts['messages'], $content);
                                            } else if
(isset($delta['content'])) {

                                                if
(isset($response["choices"][0]["delta"]["content"])) {
                                                    $text .=
$response["choices"][0]["delta"]["content"];
                                                    $isFunction = false;
                                                }
                                                if (!$dataSent) {
                                                    echo $data;
                                                    echo PHP_EOL;
                                                    ob_flush();
                                                    flush();
                                                    $dataSent = true;
                                                }
                                            } else if
(isset($response["choices"][0]['finish_reason']) &&
$response["choices"][0]['finish_reason'] == 'stop') {
                                                echo $data;
                                                $counter = 5;
                                                echo PHP_EOL;
```

```
                                        ob_flush();
                                        flush();
                                    }
                                }
                            }
                        }

                        return strlen($data);
                    });

                } catch (\Exception $exception) {
                    $counter = 5;
                    echo "data: " . $exception->getMessage();
                    echo "\n\n";
                    ob_flush();
                    flush();
                    echo 'data: [DONE]';
                    echo "\n\n";
                    ob_flush();
                    flush();
                    usleep(50000);
                }
            }
```

Then for the actual function the openai class will call

```
    function getWeather($jsonData){
        Log::info("Using weather api");
        $location = $jsonData->location;
        $url =
"http://api.weatherapi.com/v1/current.json?key=0191ce76160f4b5b9ad31403230
408&&aqi=no&q=" .urlencode($location);
        $s_response = file_get_contents($url);
        Log::info(json_encode($s_response));
        return ['role' => 'function','name' => 'get_current_weather',
'content' => $s_response];
    }
```

```
    function webSearch($jsonData){
        Log::info("Searching the web");
        $query = $jsonData->query;

        $curl = curl_init();

        curl_setopt_array($curl, array(
          CURLOPT_URL => 'https://google.serper.dev/search',
          CURLOPT_RETURNTRANSFER => true,
          CURLOPT_ENCODING => '',
          CURLOPT_MAXREDIRS => 10,
          CURLOPT_TIMEOUT => 0,
          CURLOPT_FOLLOWLOCATION => true,
          CURLOPT_HTTP_VERSION => 'CURL_HTTP_VERSION_1_1',
          CURLOPT_CUSTOMREQUEST => 'POST',
          CURLOPT_POSTFIELDS =>'{"q":"'.$query.'","num":5}',
          CURLOPT_HTTPHEADER => array(
            'X-API-KEY: 228c19b0a498a82d61554ff2801c28b4c92e0145',
            'Content-Type: application/json'
          ),
        ));

        $s_response = curl_exec($curl);
        $results = json_decode($s_response);
        $concat_results="";
        if(property_exists($results, 'knowledgeGraph')){
          $concat_results= "knowledgeGraph: " .
json_encode($results->knowledgeGraph) . "\n";
        }
        if(property_exists($results, 'answerBox')){
          $concat_results= "answerBox : " .
json_encode($results->answerBox) . "\n";
        }
        $concat_results .= "Organic:";
        foreach ($results->organic as $item) {
            $concat_results  .= ' Title: ' . $item->title . "\n";
            $concat_results  .= ' Link: ' . $item->link . "\n";
            $concat_results  .= ' Snippet: ' . $item->snippet . "\n\n";
        }
```

```
        return ['role' => 'function','name' => 'web_search', 'content' =>
$concat_results];
    }

    function advanceScraping($url){
        $ch = curl_init();
        curl_setopt($ch, CURLOPT_URL, $url);
        curl_setopt($ch, CURLOPT_PROXY,
'http://d662784298ad5c28e5200744b92c2689d7624d3f:js_render=true@proxy.zenr
ows.com:8001');
        curl_setopt($ch, CURLOPT_CUSTOMREQUEST, 'GET');
        curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
        curl_setopt($ch, CURLOPT_FOLLOWLOCATION, 1);
        curl_setopt($ch, CURLOPT_SSL_VERIFYHOST, 0);
        curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, 0);
        return curl_exec($ch);
    }

    function jsRenderingScraping($url){
        $ch = curl_init();
        curl_setopt($ch, CURLOPT_URL, $url);
        curl_setopt($ch, CURLOPT_PROXY,
'http://d662784298ad5c28e5200744b92c2689d7624d3f:js_render=true@proxy.zenr
ows.com:8001');
        curl_setopt($ch, CURLOPT_CUSTOMREQUEST, 'GET');
        curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
        curl_setopt($ch, CURLOPT_FOLLOWLOCATION, 1);
        curl_setopt($ch, CURLOPT_SSL_VERIFYHOST, 0);
        curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, 0);
        return curl_exec($ch);
    }

    function basicScraping($url){
        $ch = curl_init();
        curl_setopt($ch, CURLOPT_URL, $url);
        curl_setopt($ch, CURLOPT_PROXY,
'http://d662784298ad5c28e5200744b92c2689d7624d3f:js_render=true@proxy.zenr
ows.com:8001');
        curl_setopt($ch, CURLOPT_CUSTOMREQUEST, 'GET');
        curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
```

```
        curl_setopt($ch, CURLOPT_FOLLOWLOCATION, 1);
        curl_setopt($ch, CURLOPT_SSL_VERIFYHOST, 0);
        curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, 0);
        return curl_exec($ch);
    }

    function checkType($html){
        if (substr(trim($html), 0, 1) === '<') {
          // Probably HTML
          return 'html';
        } else if (in_array(substr(trim($html), 0, 1), ['{', '['], true))
{
            // Probably JSON
            return 'json';
        } else {
            // Unknown format
            return 'unkown';
        }
    }

    function parseHtml($html) {
        $dom = new \DOMDocument();
        // Suppress HTML errors (optional)
        libxml_use_internal_errors(true);

        // Load the HTML content into the DOMDocument
        $dom->loadHTML($html);

        // Create a DOMXPath object to navigate the DOMDocument
        $xpath = new \DOMXPath($dom);
        // Remove all script tags from the DOMDocument

        if($dom->getElementsByTagName('body')->item(0)===null){
          Log::info("no body found");
          return [false, "no body found"];
        }

        $scriptTags = $xpath->query('//script');
        foreach ($scriptTags as $scriptTag) {
            $scriptTag->parentNode->removeChild($scriptTag);
```

```
        }

        // Remove all style tags from the DOMDocument
        $styleTags = $xpath->query('//style');
        foreach ($styleTags as $styleTag) {
            $styleTag->parentNode->removeChild($styleTag);
        }

        // Get the text content of the body
        $textContent =
$dom->getElementsByTagName('body')->item(0)->textContent;

        // Clean up whitespace and remove unrelated info
        $lines = explode("\n", $textContent);
        $filteredText = '';

        foreach ($lines as $line) {
            $line = trim($line);

            // Skip empty lines and lines with just a few characters
            if (empty($line) || strlen($line) < 10) {
                continue;
            }

            // Add relevant lines to the filtered text
            // You can add additional filtering conditions based on your
specific needs
            $filteredText .= $line . "\n";
        }


        $parse_result = trim($filteredText) . "\n\n" . "Scrape Source URL:
";
        return [true, $parse_result];
    }


    function webScrape($jsonData) {
          $url = $jsonData->url;
          // Create a new DOMDocument instance
```

```
            $type = 'unkown';
            $html = null;
            $parse_result = null;


            // Basic scraping
            $html = $this->basicScraping($url);

            //if json assume it is error proceed using advanced scrape
            if($this->checkType($html)=='json'){
              $html = $this->advanceScraping($url);
            }

            $parse_result = $this->parseHtml($html);
            if(!$parse_result[0]){
              $html = $this->jsRenderingScraping($url);
              $parse_result = $this->parseHtml($html);
            }


            // Clean up whitespace and return the text content
            return ['role' => 'function','name' => 'web_scraper', 'content'
=> $parse_result[1] . $url];
        }
```

That should be the only required in the controller

Then this code reset the message content for processes like getting weather, browsing the web etc.. isProcess should be declared above