PHP AND LLMs

PRACTICAL INTEGRATIONS USING LARAVEL

The LLM hype may be fading, but these tools empower us to solve real customer needs more efficiently.

by Alfred Nutile

.001

PHP AND LLMs

Practical Integrations using Laravel

Alfred Nutile

This book is available at http://leanpub.com/php_and_llms

This version was published on 2024-08-20

The author generated this text in part with GPT-3, OpenAI's large-scale language-generation model. Upon generating draft language, the author reviewed, edited, and revised the language to their own liking and takes ultimate responsibility for the content of this publication.

Tweet This Book!

Please help Alfred Nutile by spreading the word about this book on Twitter!

The suggested tweet for this book is:

Excited to learn practical applications of LLMs in my Laravel applications.

The suggested hashtag for this book is #laravelandllms.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

#laravelandllms

Contents

[COMPLETED] Introduction - Embracing the Reality of LLMs in Development

Yes, the hype around LLMs might be fading, but the reality is that these tools offer transformative possibilities for developers. My goal is to provide you with the knowledge and confidence to integrate LLMs into your projects, work effectively with them locally, and add a powerful new tool to your development arsenal. And we won't just skim the surface with "Hello World" examples. We're going to dig deep into the real-world applications that can genuinely enhance your day-to-day work as a developer.

> PHP has a place in this seemingly Python + Ai era.

By the end of this book, I'll equip you with the skills to:

- Work Locally with Ollama: Learn how to save costs and iterate on your ideas effectively by working locally with Ollama. This approach will allow you to refine your concepts without the overhead of cloud services, giving you the flexibility to develop and test in your own environment.
- Test and Mock with Confidence: We'll explore how to set up robust testing and mocking frameworks, so you can deploy your code with the assurance that it will perform reliably in production. This chapter will help you build a foundation of confidence, knowing that your integrations are solid and ready for real-world use.
- Integrate Multiple LLMs Seamlessly: Discover how to write code that's flexible enough to work with any LLM service—whether it's Claude, Ollama, OpenAI, Groq, or another. We'll cover strategies for creating a unified interface, allowing you to switch between services or use multiple providers simultaneously, all within the same codebase.
- Apply Real-World Solutions: I'll share numerous examples of real-world solutions I've implemented (or plan to implement) that demonstrate the practical power of LLMs. These examples will show you how to save time, write efficient code, and tackle complex situations that might have seemed daunting before.
- Master Tools for LLMs: One of the most critical skills you'll learn is how to empower your prompts with tools. We'll dive into how tools can drive a wide range of tasks and automations, making your LLMs not just smart, but practically useful in day-to-day development.
- Make Tools LLM-Agnostic: Finally, we'll cover how to create tools that are independent of any specific LLM. This means you can write your prompts and logic once and use them across multiple LLMs, ensuring that your work remains flexible and future-proof.

By the end of this book, you'll not only have the skills and confidence to move forward with LLM technology but also a clear understanding of how to apply it practically in your work. This journey is about more than just learning a new tool; it's about expanding your capabilities as a developer and seeing the real potential that LLMs bring to solving complex problems efficiently and effectively.

Local LLMs Streamlining Your Development Workflow

Summary: We explore the benefits of working with local LLMs like Ollama, which allow you to develop and test your ideas without incurring cloud costs. This chapter includes practical tips for setting up local environments and integrating them into your workflow. This will include tips and tricks on how to test locally and in CI.

Building the LLM Agnostic Driver

Summary: This chapter covers how to integrate multiple LLMs into your projects, allowing you to switch between services like OpenAI, Claude, and Groq seamlessly. We discuss the advantages of this approach and provide examples of how to implement it effectively.

Prompting Overview before we Dig In

Summary: This chapter delves into the importance of crafting the right prompts and providing the correct context to LLMs. We explore different prompt iterations and demonstrate how to refine them to get the best results, focusing on a real-world example of parsing opt-out emails.

[COMPLETED] Parsing Opt-Out Text

Summary of What You Will Learn

- Understand how to use LLMs for parsing unstructured text data like opt-out emails.
- Learn the importance of tuning prompts and providing the right context to LLMs.
- See how LLM-based parsing can reduce code complexity and manual effort.
- Gain insights into handling variations in data formats and extracting specific details.

The Goal

So far, we've covered how to connect to LLMs, pass a request, and get a response.

In this chapter, I'll walk you through a problem I had to solve for a customer that perfectly showcases the power of these tools. The customer receives emails from various opt-out providers requesting to remove individuals from their mailing list. The catch? The data comes in all sorts of formats—some not even plain text, but just HTML!

In the past, I would have written a text parser specific to each domain to extract the Full Name, Phone Number, Address, and Reply-To Address (which, by the way, isn't always in the email header).

But one day, when a new domain started sending us requests, I had a lightbulb moment: why not use an LLM to handle the parsing?

First and foremost, you need to spend some time fine-tuning the prompt that you will pass to the llm with the email content. Ultimately, everything boils down to getting the right Prompt (text) and the right Context (text) to the LLM. This principle holds true no matter what problem you're trying to solve.

> It's all about getting the right Prompt (text) and the right Context (text) to the LLM.

With this in mind, using the Driver system we built in a previous chapter, we can plug this into Tinkerwell and start testing our ideas.

```php
<?php
$results = LlmDriverFacade::driver("ollama")
  ->setFormat("json")
  ->completion($prompt);

$results->content;
```

> Note we are doing `setFormat` since many of these LLMs accept that. We then, depending on the LLM alter the request as needed.

The response we get is:

```
{ "full_name": "John Doe", \n
    "home_address": "123 Main Street, Anytown, USA", \n
    "other_name": null, \n
    "reply_to": "johndoe@example.com", \n
    "valid_user_email": "johndoe@example.com", \n
    "phone": "555-555-5555"\n
}
```

You will notice that in the sample email below, parsing this text is possible BUT to write this for every opt-out vendor, some which had harder to parse emails, it was more work than to do this once.

$email

```
Subject: Request for Deletion of Personal Information

My name is John Doe, and I request that Your Company deletes all of the information \
it has collected about me, whether directly from me, through a third party, or throu
gh a service provider.

The following information is provided as required for validation of my request:

First Name: John
Last Name: Doe
Email: johndoe@example.com
Address: 123 Main Street, Anytown, USA
Phone Number: 555-555-5555

If you need any more information, please let me know as soon as possible. If you can\
not comply with my request—either in whole or in part—please state the reason why yo
u cannot comply. If part of my information is subject to an exception, please delete
```

```
 all information that is not subject to an exception. If my request is incomplete, p
lease provide me with specific instructions on how to complete my request.

I have used a consumer identity protection service to send this email, but please re\
spond directly to my personal email address johndoe@example.com.

I am sending this request exercising my rights to my Personal Data under applicable \
privacy laws, including, but not limited to, the General Data Protection Regulation
(“GDPR”) and the California Consumer Privacy Act (“CCPA”)
```

$prompt

```
$prompt = <<<PROMPT
<role>
Help parse opt out emails so we can remove the person from our system or ignore the \
email if it is
not related to opt out.

<task>
Review the email below and then if it is an opt out request pull the data out as def\
ined by the format section.
valid_user_email might be tricky since sometimes the email is generatic and going ba\
ck to the company sending these opt outs.
But if you are confident the email looks like a valid personal or business email add\
 that to valid_user_email.

<format>
JSON FORMAT with key values below, using null if no value for the key. This is valid\
 JSON not markdown.
If there is no data in this about a person just return the JSON below with all field\
s null. No surrounding text.

{
 "full_name": null,
 "home_address": null,
 "other_name": null,
 "reply_to": null,
 "valid_user_email": null,
 "phone": null
}

Only return valid JSON

```

```
{$email}
PROMPT;
```

> NOTE this is like mu 4-5 th prompt attempt, "valid JSON" phrase came from the the www.deeplearning.ai trainings.

Then we pass that to the LlmDriver:

```
<?php
$results = LlmDriverFacade::driver("claude")->completion($prompt);
$results->content;
```

> Note we will swap out drivers to show how it works.

And we still get:

```
{\n
   "full_name": "John Doe",\n
   "home_address": "123 Main Street, Anytown, USA",\n
   "other_name": null,\n
   "reply_to": null,\n
   "valid_user_email": "johndoe@example.com",\n
   "phone": "555-555-5555"\n
  }
```

As you can see, we use the driver, which can be any LLM—whether it's "ollama", "claude," "openai," or "groq." We then pass it the prompt along with the HTML/Text of the email.

We also enforce a JSON format for the output, which gives us some confidence in the results. From here, we can decide how to proceed:

- Ignore the output if all the fields are null.
- Process the output if the required fields are present.

Also, note that the reply-to address is embedded in the email. Many of these opt-outs require us to reply to the email that sent the request, but as you can see here, we had to adapt to this one.

This approach saved me from manually parsing the text out of these emails and significantly reduced the amount of code, as I previously had parsers for two other vendors.

Later I will talk abouy how to evaluate the accuracy of the results, as well as ideas on testing.

What You Have Learned

- You've learned how to leverage LLMs to parse unstructured text data, such as opt-out emails.
- You understand the importance of crafting effective prompts and providing relevant context to LLMs.
- You've explored techniques for extracting specific information from different email formats.
- You can see how LLM-based parsing simplifies text processing and saves development time.
- You've gained valuable insights into handling data variations and extracting key details from emails.

Tools Overview Enhancing LLM Capabilities

Summary: This chapter introduces the concept of tools that enhance the functionality of LLMs. We explore how to build tools that can work with any LLM and how to integrate them into your projects. A real-life example of an event scraping tool is used to demonstrate these concepts.

[COMPLETED] - Web Scraping and Building an Events Database

Summary of What You Will Learn

- Understand how to process data scraped from websites using LLMs.
- Learn to create tools that can manage and process complex arrays of objects like events.
- Explore the use of LLMs to find and create events from web pages.
- See how to define complex parameter structures (JSON) for LLM tools.
- Gain insights into orchestration, chaining tool results, and generating a final response.

Big Picture View

Here, we explore how we can take data scraped from a website and use LLMs to process the content. The chapter includes examples of tools that can extract relevant information from web pages and convert it into Events in our database.

We don't delve into scraping here, since it's pretty common and there are many ways to do it. We focus on building a tool that can handle a complex array of objects (Events) for the LLM to pass into it.

Once we have that tool in place, the following prompt can do a lot:

```
<role>
You are an assistant helping to parse data from HTML content and find the events on \
the page.
<task>
Using the context below, find all the events and pass them into the create event too\
l.
After that, save it to a document.
<context>
[INSERT OUR HTML HERE]
```

As you can see above, we tell the prompt to use our tool. Keep in mind, users may not be so specific; they might write "Find and create events from this page", but we can test that as well.

We need to create a simple Event model in the database to store the events. After that, we'll build the tool and define the necessary arguments. When the LLM provides the function call, the tool will then generate the events.

As we saw in a previous chapter, Tools are ways to define a function and parameters that the LLM can use to complete a task. The task can be simple like `get_the_weather`, but we are going to do more complex tasks. To start with, this tool is going to have a complex parameter definition (this will be JSON for Claude; we'll show the abstracted version later in this chapter).

The `get_the_weather` is a Class we made to process the arguments. Below is the example structure for the `create_event_tool`

```
{
    "name": "create_event_tool",
    "description": "If the user needs to create one or more events this tool helps",
    "input_schema": {
        "type": "object",
        "properties": {
            "events": {
                "description": "Array of event objects",
                "type": "array",
                "items": {
                    "type": "object",
                    "properties": {
                        "start_time": {
                            "type": "string",
                            "description": "Start time of the event"
                        },
                        "end_time": {
                            "type": "string",
                            "description": "End time of the event"
                        },
                        "title": {
                            "type": "string",
                            "description": "Title of the event"
                        },
                        "location": {
                            "type": "string",
                            "description": "Location of the event"
                        },
                        "summary": {
                            "type": "string",
                            "description": "Summary or description of the event"
                        }
```

```
                    }
                }
            }
        },
        "required": [
            "events"
        ]
    }
}
```

This took me a while to get right, but thanks to Claude's help, I was able to figure out how to do an array of objects.

- name - this will be seen later registered in our application.
- description - here we do a good job of letting the LLM know how to use the tool.
- input_schema - this is the tricky part

You can see here a very simple tool in the ChatGPT docs[1] as an example :

```
{
    "name": "get_weather",
    "description": "Get the current weather in a given location",
    "input_schema": {
      "type": "object",
      "properties": {
        "location": {
          "type": "string",
          "description": "The city and state, e.g. San Francisco, CA"
        }
      },
      "required": ["location"]
    }
}
```

Now, let's take it up a notch. Our first property "events" has the type of "array".

And in that object key, we start to define our properties. As you can see in the `curl` command below, we are just going to pass this to Claude.

[1] https://platform.openai.com/docs/guides/structured-outputs/all-fields-must-be-required

```
curl --location 'https://api.anthropic.com/v1/messages' \
--header 'x-api-key: my_key' \
--header 'anthropic-version: 2023-06-01' \
--header 'content-type: application/json' \
--data '{
    "model": "claude-3-5-sonnet-20240620",
    "max_tokens": 4096,
    "messages": [
        {
            "role": "user",
            "content": "ALL THE HTML / TEXT WITH EVENTS"
        }
    ],
    "tools": [
    {
        "name": "create_event_tool",
        "description": "If the user needs to create one or more events this tool hel\
ps",
        "input_schema": {
            "type": "object",
            "properties": {
                "events": {
                    "description": "Array of event objects",
                    "type": "array",
                    "items": {
                        "type": "object",
                        "properties": {
                            "start_time": {
                                "type": "string",
                                "description": "Start time of the event"
                            },
                            "end_time": {
                                "type": "string",
                                "description": "End time of the event"
                            },
                            "title": {
                                "type": "string",
                                "description": "Title of the event"
                            },
                            "location": {
                                "type": "string",
                                "description": "Location of the event"
                            },
```

```
                            "summary": {
                                "type": "string",
                                "description": "Summary or description of the event"
                            }
                        }
                    }
                }
            },
            "required": [
                "events"
            ]
        }
    }
]
}'
```

It will reply as follows:

```
{
    "id": "msg_01GDtWoCBAnDpdLmLbJZGp2u",
    "type": "message",
    "role": "assistant",
    "model": "claude-3-5-sonnet-20240620",
    "content": [
        {
            "type": "text",
            "text": "Certainly! I'll parse the events from the Dallas Cowboys schedu\
le and create them using the create_event_tool. I'll focus on the regular season gam
es, as there are quite a few events. Let's create these events:"
        },
        {
            "type": "tool_use",
            "id": "toolu_01B9W2pzR5RqDj11QciiseJq",
            "name": "create_event_tool",
            "input": {
                "events": [
                    {
                        "title": "Dallas Cowboys at Cleveland Browns",
                        "start_time": "2024-09-08T15:25:00-05:00",
                        "end_time": "2024-09-08T18:25:00-05:00",
                        "location": "Cleveland Browns Stadium",
                        "summary": "NFL Week 1: Dallas Cowboys at Cleveland Browns"
                    },
```

```
                    {
                        "title": "Dallas Cowboys vs New Orleans Saints",
                        "start_time": "2024-09-15T12:00:00-05:00",
                        "end_time": "2024-09-15T15:00:00-05:00",
                        "location": "AT&T Stadium",
                        "summary": "NFL Week 2: Dallas Cowboys vs New Orleans Saints"
                    },
                    {
                        "title": "Dallas Cowboys vs Baltimore Ravens",
                        "start_time": "2024-09-22T15:25:00-05:00",
                        "end_time": "2024-09-22T18:25:00-05:00",
                        "location": "AT&T Stadium",
                        "summary": "NFL Week 3: Dallas Cowboys vs Baltimore Ravens"
                    },
                    {
                        "title": "Dallas Cowboys at New York Giants",
                        "start_time": "2024-09-26T19:15:00-05:00",
                        "end_time": "2024-09-26T22:15:00-05:00",
                        "location": "MetLife Stadium",
                        "summary": "NFL Week 4: Dallas Cowboys at New York Giants"
                    },
                    {
                        "title": "Dallas Cowboys at Pittsburgh Steelers",
                        "start_time": "2024-10-06T19:20:00-05:00",
                        "end_time": "2024-10-06T22:20:00-05:00",
                        "location": "Acrisure Stadium",
                        "summary": "NFL Week 5: Dallas Cowboys at Pittsburgh Steeler\
s"
                    }
                ]
            }
        }
    ],
    "stop_reason": "tool_use",
    "stop_sequence": null,
    "usage": {
        "input_tokens": 2652,
        "output_tokens": 593
    }
}
```


Our tool will iterate over these results to create events—simple, right? But wait, there's a catch! Keep reading!

Orchestration

It is really easy to call the LLM with PHP code like:

```php
<?php

$result = $client->chat()->create([
    'model' => 'gpt-4',
    'messages' => [
        ['role' => 'user', 'content' => 'Hello!'],
    ],
]);
```

However, we want to do more than that: we want to be able to chain the results of that to either one or more tools and a final response or just the final response. That is where an Orchestration-like class comes in.

```php
<?php
public function handle(Chat $chat, string $prompt): Message
{
    $chat->addInput(
        message: $prompt,
        role: RoleEnum::User
    );

    $messageInDto = MessageInDto::from([
        'content' => $prompt,
        'role' => 'user',
    ]);

    $response = LlmDriverFacade::driver($chat->getDriver())
        ->chat([
            $messageInDto,
        ]);

    if (! empty($response->tool_calls)) {
        Log::info('Orchestration Tools Found', [
            'tool_calls' => collect($response->tool_calls)
                ->pluck('name')->toArray(),
        ]);

        $count = 1;
```

```
26          foreach ($response->tool_calls as $tool_call) {
27              Log::info('[LaraChain] - Tool Call '.$count, [
28                  'tool_call' => $tool_call->name,
29                  'tool_count' => count($response->tool_calls),
30              ]);
31
32              $message = $chat->addInput(
33                  message: $response->content ?? 'Calling Tools', //@NOTE ollama, open\
34  ai blank but claude needs this :(
35                  role: RoleEnum::Assistant,
36                  tool: $tool_call->name,
37                  tool_id: $tool_call->id,
38                  args: $tool_call->arguments,
39              );
40
41              $tool = app()->make($tool_call->name);
42              $results = $tool->handle($message);
43              $message->updateQuietly([
44                  'role' => RoleEnum::Tool,
45                  'body' => $results->content,
46              ]);
47              $count++;
48          }
49
50          $messages = $chat->getChatResponse();
51
52          $response = LlmDriverFacade::driver($chat->getDriver())
53              ->chat($messages);
54
55          return $chat->addInput(
56              message: $response->content,
57              role: RoleEnum::Assistant,
58          );
59
60      } else {
61          Log::info('No Tools found just gonna chat');
62          $assistantMessage = $chat->addInput(
63              message: $response->content ?? 'Calling Tools',
64              role: RoleEnum::Assistant
65          );
66
67          return $assistantMessage;
68      }
```

```
69  }
```

As we discussed in earlier chapters, it's important to maintain the state of the chat and messages. To achieve this, I typically use a Chat and Message model. This setup allows me to seamlessly connect everything based on the maintained state. We covered this in a previous chapter.

Let's look at this line by line:

```php
<?php
$messages = $chat->getChatResponse();
```

Here, we tell the Chat model to give us the thread of messages so we can continue our chat with the LLM.

```php
<?php
$response = LlmDriverFacade::driver($message->getDriver())
    ->chat($messages);
```

We pass our message to the LLMs. But, in here is the LLM Driver loading the tools we have.

```php
<?php
$this->getFunctions();
```

We will look at this one in more detail shortly, but just consider it will load the tool we are going to make, register it with our Driver so it knows to offer this tool as an option in the chat interaction.

Let's zoom into chat with tools:

```php
<?php
foreach ($response->tool_calls as $tool_call) {
    $message = $chat->addInput(
        message: $response->content ?? 'Calling Tools', //ollama, openai blank but c\
laude needs this :(
        role: RoleEnum::Assistant,
        show_in_thread: false,
        meta_data: MetaDataDto::from([
            'tool' => $tool_call->name,
            'tool_id' => $tool_call->id,
            'driver' => $chat->getDriver(),
            'args' => $tool_call->arguments,
        ]),
    );
    $tool = app()->make($tool_call->name);
```

```
    $results = $tool->handle($message);
    $message->updateQuietly([
        'is_chat_ignored' => true,
        'role' => RoleEnum::Tool,
        'body' => $results->content,
    ]);
    $count++;
}
```

You can see it just iterates over tool(s) until it is done, then finishes off with:

```php
<?php
$messages = $chat->getChatResponse();

/**
 * @NOTE
 * I have to continue to pass in tools once used above
 * Since Claude needs them.
 */
$response = LlmDriverFacade::driver($chat->getDriver())
    ->setToolType(ToolTypes::Source)
    ->chat($messages);

$assistantMessage = $chat->addInput(
    message: $response->content,
    role: RoleEnum::Assistant,
    show_in_thread: true,
    meta_data: null,
    tools: null
);
```

So as you see above, we use `getChatResponse` to get the message thread, then we pass the final results after all the tools are done being called, back to the LLM to make the final message.

Deep Dive

Ok now that we are done with the "Big Picture View", let's dive into the details.

Start with the Request

The request comes in via a Job, Controller, or Command (there are other ways but just to focus on a few). The request then does the work, in this case to scrape a webpage. Then, we take that content and pass it into the `Orchestrate` class like this:

https://github.com/alnutile/php-llms/blob/main/app/Services/LlmServices/Orchestration/Orchestrate.php#L14

```php
<?php
use Facades\App\Services\LlmServices\Orchestration\Orchestrate;
Orchestrate::handle($chat, $prompt);
```

NOTE: These classes have tests if you want to read how they work

So what is happening here is we are always dealing with Chat. I do this so we have a root to the thread. Sometimes we will use Message, but in this case, we are working at the Chat level. You will see in a prior chapter we made the Chat model.

And the `$prompt` is the start of this work, which by now has gone through a few revisions. The prompt is basically the question the user is asking of the system. The question has a tool in it by happenstance in that the user might not even know the tool exists; they just asked for something that happened to have the needed tools.

The Orchestrate class then hands the question and the tools to the LLM to see if it can answer it with or without tools.

If the response comes back tool-based, e.g.:

```
{
    "id": "msg_01GDtWoCBAnDpdLmLbJZGp2u",
    "type": "message",
    "role": "assistant",
    "model": "claude-3-5-sonnet-20240620",
    "content": [
        {
            "type": "text",
            "text": "Certainly! I'll parse the events from the Dallas Cowboys schedu\
le and create them using the create_event_tool. I'll focus on the regular season gam
es, as there are quite a few events. Let's create these events:"
        },
        {
            "type": "tool_use",
            "id": "toolu_01B9W2pzR5RqDj11QciiseJq",
            "name": "create_event_tool",
            "input": {
                "events": [
                    {
                        "title": "Dallas Cowboys at Cleveland Browns",
                        "start_time": "2024-09-08T15:25:00-05:00",
```

```
                        "end_time": "2024-09-08T18:25:00-05:00",
                        "location": "Cleveland Browns Stadium",
                        "summary": "NFL Week 1: Dallas Cowboys at Cleveland Browns"
                    },
                    {
                        "title": "Dallas Cowboys vs New Orleans Saints",
                        "start_time": "2024-09-15T12:00:00-05:00",
                        "end_time": "2024-09-15T15:00:00-05:00",
                        "location": "AT&T Stadium",
                        "summary": "NFL Week 2: Dallas Cowboys vs New Orleans Saints"
                    },
                    {
                        "title": "Dallas Cowboys vs Baltimore Ravens",
                        "start_time": "2024-09-22T15:25:00-05:00",
                        "end_time": "2024-09-22T18:25:00-05:00",
                        "location": "AT&T Stadium",
                        "summary": "NFL Week 3: Dallas Cowboys vs Baltimore Ravens"
                    },
                    {
                        "title": "Dallas Cowboys at New York Giants",
                        "start_time": "2024-09-26T19:15:00-05:00",
                        "end_time": "2024-09-26T22:15:00-05:00",
                        "location": "MetLife Stadium",
                        "summary": "NFL Week 4: Dallas Cowboys at New York Giants"
                    },
                    {
                        "title": "Dallas Cowboys at Pittsburgh Steelers",
                        "start_time": "2024-10-06T19:20:00-05:00",
                        "end_time": "2024-10-06T22:20:00-05:00",
                        "location": "Acrisure Stadium",
                        "summary": "NFL Week 5: Dallas Cowboys at Pittsburgh Steeler\
s"
                    }
                ]
            }
        }
    ],
    "stop_reason": "tool_use",
    "stop_sequence": null,
    "usage": {
        "input_tokens": 2652,
        "output_tokens": 593
    }
```

```
65 }
```

Then we process the tools requests.

The LLM Client

All of this happens in the Client, we will focus on the Claude client for now. https://github.com/alnutile/php-llms/blob/main/app/Services/LlmServices/ClaudeClient.php

> We cover this in an earlier chapter

```php
<?php
$response = LlmDriverFacade::driver($chat->getDriver())
    ->chat([
        $messageInDto,
    ]);
```

The driver is set to `claude` so we then talk to the CluadeClient chat method:

```php
<?php
/**
 * @param  MessageInDto[]  $messages
 */
public function chat(array $messages): CompletionResponse
{
    $model = $this->getConfig('claude')['models']['completion_model'];
    $maxTokens = $this->getConfig('claude')['max_tokens'];

    Log::info('LlmDriver::Claude::chat');

    $messages = $this->remapMessages($messages);

    $payload = [
        'model' => $model,
        'max_tokens' => $maxTokens,
        'messages' => $messages,
    ];

    $payload = $this->modifyPayload($payload);

    $results = $this->getClient()->post('/messages', $payload);
```

```
23
24      if (! $results->ok()) {
25          $error = $results->json()['error']['type'];
26          $message = $results->json()['error']['message'];
27          Log::error('Claude API Error Chat', [
28              'type' => $error,
29              'message' => $message,
30          ]);
31
32          throw new \Exception('Claude API Error Chat');
33      }
34
35      return ClaudeCompletionResponse::from($results->json());
36  }
```

We cover a lot of this in other chapters, but for now I want to focus on:

```
1  <?php
2  $payload = $this->modifyPayload($payload);
```

This is how we turned the `CreateEventTool` into an LLM agnostic tool payload.

The `modifyPayload` calls the parent class to get the functions and then its own `remapFunctions` (seen below) all the LLM client have their own remapping, though most use the same format as ChatGPT.

The code below does not matter as much as realizing that we are taking the formatting of the tools parameters (will show after this) and turning them into the JSON format Claude API requires.

```
1  <?php
2  public function remapFunctions(array $functions): array
3      {
4          return collect($functions)->map(function ($function) {
5              $properties = [];
6              $required = [];
7
8              $type = data_get($function, 'parameters.type', 'object');
9
10             foreach (data_get($function, 'parameters.properties', []) as $property) {
11                 $name = data_get($property, 'name');
12
13                 if (data_get($property, 'required', false)) {
14                     $required[] = $name;
15                 }
16
```

```
                $subType = data_get($property, 'type', 'string');
                $properties[$name] = [
                    'description' => data_get($property, 'description', null),
                    'type' => $subType,
                ];

                if ($subType === 'array') {
                    $subItems = $property['properties'][0]->properties; //stop at th\
is for now
                    $subItemsMapped = [];
                    foreach ($subItems as $subItemKey => $subItemValue) {
                        $subItemsMapped[$subItemValue->name] = [
                            'type' => $subItemValue->type,
                            'description' => $subItemValue->description,
                        ];
                    }

                    $properties[$name]['items'] = [
                        'type' => 'object',
                        'properties' => $subItemsMapped,
                    ];
                }
            }

            $itemsOrProperties = $properties;

            return [
                'name' => data_get($function, 'name'),
                'description' => data_get($function, 'description'),
                'input_schema' => [
                    'type' => 'object',
                    'properties' => $itemsOrProperties,
                    'required' => $required,
                ],
            ];
        })->values()->toArray();
    }
```

At this point we just make an HTTP request to the API and return it.

> NOTE: I talk about `MessageInDto` and `ClaudeCompletionResponse` in another section but the goal here is ALL llms take the same Data Object and return the same data object. In this case `ClaudeCompletionResponse` extends `CompletionResponse`. This is

really powerful in that we can now talk to and return the same object from any LLM we plug in.

The CrateEventTool

Ok now that we see how Orchestrate works and how the ClaudeClient reformats the functions lets look at the foundation to tools and in this case the `CreateEventTool`

https://github.com/alnutile/php-llms/blob/main/app/Services/LlmServices/Functions/CreateEventTool.php

You will see this extends `FunctionContract` so let's start with that below:

https://github.com/alnutile/php-llms/blob/main/app/Services/LlmServices/Functions/FunctionContract.php

```php
<?php

namespace App\Services\LlmServices\Functions;

use App\Models\Message;

abstract class FunctionContract
{
    protected string $name;

    protected string $description;

    protected string $type = 'object';

    abstract public function handle(
        Message $message,
    ): FunctionResponse;

    public function getFunction(): FunctionDto
    {
        return FunctionDto::from(
            [
                'name' => $this->getName(),
                'description' => $this->getDescription(),
                'parameters' => [
                    'type' => $this->type,
                    'properties' => $this->getProperties(),
```

```
                ],
            ]
        );
    }

    public function getName(): string
    {
        return $this->name;
    }

    public function getKey(): string
    {
        return $this->name;
    }

    public function getDescription(): string
    {
        return $this->description;
    }

    public function getParameters(): array
    {
        return $this->getProperties();
    }

    /**
     * @return PropertyDto[]
     */
    abstract protected function getProperties(): array;
}
```

Our tools must have a `name` and a good `description`.

> See some docs at https://docs.anthropic.com/en/docs/build-with-claude/tool-use

Then, we have the `handle` that all tools have that will return the object `FunctionDto`

```php
<?php

namespace App\Services\LlmServices\Functions;

class FunctionDto extends \Spatie\LaravelData\Data
{
    public function __construct(
        public string $name,
        public string $description,
        public ParametersDto $parameters,
    ) {}
}
```

Ok now lets see how the `CreateEventTool` extends this:

```php
<?php

namespace App\Services\LlmServices\Functions;

use App\Models\Event;
use App\Models\Message;
use Carbon\Carbon;
use Illuminate\Support\Facades\Log;

class CreateEventTool extends FunctionContract
{
    protected string $name = 'create_event_tool';

    protected string $description = 'If the user needs to create one or more events \
this tool help';

    public function handle(
        Message $message): FunctionResponse
    {
        Log::info('CreateEventTool called');

        $args = $message->args;

        $eventArray = data_get($args, 'events', []);

        foreach ($eventArray as $event) {
            $start_date = null;
            $end_date = null;
```

```php
            $description = data_get($event, 'description', null);
            $start_time = data_get($event, 'start_time', null);
            $end_time = data_get($event, 'end_time', null);
            $location = data_get($event, 'location', null);
            $title = data_get($event, 'title', 'No Title Found');
            $all_day = data_get($event, 'all_day', false);

            if ($start_time != '' || $start_time != null) {
                $start_date = Carbon::parse($start_time)->format('Y-m-d');
                $start_time = Carbon::parse($start_time)->format('H:i:s');
            }

            if ($end_time != '' || $end_time != null) {
                $end_date = Carbon::parse($end_time)->format('Y-m-d');
                $end_time = Carbon::parse($end_time)->format('H:i:s');
            }

            Event::updateOrCreate([
                'title' => $title,
                'start_date' => $start_date,
                'start_time' => $start_time,
                'location' => $location,
            ],
                [
                    'description' => $description,
                    'end_date' => $end_date,
                    'end_time' => $end_time,
                    'all_day' => $all_day,
                ]);
        }

        return FunctionResponse::from([
            'content' => json_encode($eventArray),
            'prompt' => $message->body,
        ]);
    }

    /**
     * @return PropertyDto[]
     */
    protected function getProperties(): array
    {
        return [
```

```
            new PropertyDto(
                name: 'events',
                description: 'Array of event objects',
                type: 'array',
                required: true,
                properties: [
                    new PropertyDto(
                        name: 'items',
                        description: 'Event object',
                        type: 'object',
                        required: true,
                        properties: [
                            new PropertyDto(
                                name: 'start_time',
                                description: 'Start time of the event',
                                type: 'string',
                                required: true
                            ),
                            new PropertyDto(
                                name: 'end_time',
                                description: 'End time of the event',
                                type: 'string',
                                required: false
                            ),
                            new PropertyDto(
                                name: 'title',
                                description: 'Title of the event',
                                type: 'string',
                                required: true
                            ),
                            new PropertyDto(
                                name: 'location',
                                description: 'Location of the event',
                                type: 'string',
                                required: false
                            ),
                            new PropertyDto(
                                name: 'description',
                                description: 'Description of the event',
                                type: 'string',
                                required: true
                            ),
                        ]
```

```
                    ),
                ]
            ),
        ];
    }
}
```

Let's look first at the function/tool name `create_event_tool` this is key since we register this in https://github.com/alnutile/php-llms/blob/main/app/Services/LlmServices/LlmServiceProvider.php which we registered in an earlier chapter.

```php
<?php
    public function boot(): void
    {
        $this->app->bind('llm_driver', function () {
            return new LlmDriverClient;
        });

        $this->app->bind('create_event_tool', function () {
            return new CreateEventTool;
        });

    }
```

Then we register the tool in the `BaseClient`: https://github.com/alnutile/php-llms/blob/main/app/Services/LlmServices/BaseClient.php

```php
<?php
    public function getFunctions(): array
    {
        $functions = collect(
            [
                new CreateEventTool,
            ]
        );

        return $functions->transform(
            /** @phpstan-ignore-next-line */
            function (FunctionContract $function) {
                return $function->getFunction();
            }
        )->toArray();
    }
```

So, now when any LLM Client runs and calls `getFunctions` seen above, it will get the needed functions from this method and we append that to the chat payload.

```
<?php
//app/Services/LlmServices/Orchestration/Orchestrate.php
foreach ($response->tool_calls as $tool_call) {
    Log::info('[LaraChain] - Tool Call '.$count, [
        'tool_call' => $tool_call->name,
        'tool_count' => count($response->tool_calls),
    ]);

    $message = $chat->addInput(
        message: $response->content ?? 'Calling Tools', //@NOTE ollama, openai blank\
 but claude needs this :(
        role: RoleEnum::Assistant,
        tool: $tool_call->name,
        tool_id: $tool_call->id,
        args: $tool_call->arguments,
    );

    $tool = app()->make($tool_call->name);
    $results = $tool->handle($message);
    $message->updateQuietly([
        'role' => RoleEnum::Tool,
        'body' => $results->content,
    ]);
    $count++;
}
```

`Orchestrate` then knows to call the tool class and its handle method.

> Note that `Orchestrate` sets the `$message` to have `args` and other tool info. That is key.

At this point, we see that the tool gets the args and iterates over them, knowing the format will match its parameters.

```php
<?php
$eventArray = data_get($args, 'events', []);

foreach ($eventArray as $event) {
    $start_date = null;
    $end_date = null;
    $description = data_get($event, 'description', null);
    $start_time = data_get($event, 'start_time', null);
    $end_time = data_get($event, 'end_time', null);
    $location = data_get($event, 'location', null);
    $title = data_get($event, 'title', 'No Title Found');
    $all_day = data_get($event, 'all_day', false);

    if ($start_time != '' || $start_time != null) {
        $start_date = Carbon::parse($start_time)->format('Y-m-d');
        $start_time = Carbon::parse($start_time)->format('H:i:s');
    }

    if ($end_time != '' || $end_time != null) {
        $end_date = Carbon::parse($end_time)->format('Y-m-d');
        $end_time = Carbon::parse($end_time)->format('H:i:s');
    }

    Event::updateOrCreate([
        'title' => $title,
        'start_date' => $start_date,
        'start_time' => $start_time,
        'location' => $location,
    ],
        [
            'description' => $description,
            'end_date' => $end_date,
            'end_time' => $end_time,
            'all_day' => $all_day,
    ]);

    return FunctionResponse::from([
        'content' => json_encode($eventArray),
        'prompt' => $message->body,
    ]);
}
```

Finally, all Tools return `FunctionResponse` so `Orchestrate` can use the response if needed.

Let's look at the `getProperties` of the `CreateEventTool`. As noted in a previous chapter, we abstract out the parameters so that it starts as and array of `PropertyDto` objects. That later any LLM can transform.

```php
<?php
/**
 * @return PropertyDto[]
 */
protected function getProperties(): array
{
    return [
        new PropertyDto(
            name: 'events',
            description: 'Array of event objects',
            type: 'array',
            required: true,
            properties: [
                new PropertyDto(
                    name: 'items',
                    description: 'Event object',
                    type: 'object',
                    required: true,
                    properties: [
                        new PropertyDto(
                            name: 'start_time',
                            description: 'Start time of the event',
                            type: 'string',
                            required: true
                        ),
                        new PropertyDto(
                            name: 'end_time',
                            description: 'End time of the event',
                            type: 'string',
                            required: false
                        ),
                        new PropertyDto(
                            name: 'title',
                            description: 'Title of the event',
                            type: 'string',
                            required: true
                        ),
                        new PropertyDto(
                            name: 'location',
```

```
                            description: 'Location of the event',
                            type: 'string',
                            required: false
                        ),
                        new PropertyDto(
                            name: 'description',
                            description: 'Description of the event',
                            type: 'string',
                            required: true
                        ),
                    ]
                ),
            ]
        ),
    ];
}
```

What You've Learned

In summary, we see a few patterns that lay the foundation to all our interactiosn with LLMs (some of this we covered in previous chapters):

- Our LLM Clients have a DataObject in and DataObject out that they all share.
- Our Tools have parameters that are objects that then any LLM Client can transform to the needed JSON for that API.
- The messages we get back have to be transformed as well and like above the different LLM Clients need a different format to the messages.
- Tools are powerful and with the Orchestrate class we can chain them together naturally or forced (more on that later)

Making Tools LLM-Agnostic

Summary: Building on the previous chapter, we focus on creating tools that are not tied to any specific LLM. This allows you to switch between LLMs like OpenAI, Ollama, and Claude without rewriting your code. The chapter includes examples of how to structure your tools to be flexible and adaptable.

Chaining Tools

Summary: This chapter explores how to iterate over tools in complex workflows. We discuss real-world scenarios where multiple tools need to be chained together to achieve a final result. Examples include processing web content and generating images, highlighting the importance of tool chaining.

Web Searches with LLMs

Summary: A case study of automating web searches using LLMs. The chapter walks through setting up a system that regularly searches the web for relevant content, processes the results, and takes further actions based on predefined criteria.a

Marketing Assistant

Summary:

This tool will help us make events via a prompt about helping us with marketing. This will allow the system to then track the status of those events, let us know what is coming up, and ideally automate some of the actions of those events.a

Hand Written Notes to Task List

Summary: In this chapter, we explore how to leverage LLMs to transform handwritten notes and voice files into actionable task lists. We'll discuss the process of uploading these inputs, using LLMs to interpret and organize the content, and generating structured tasks that can be integrated into your workflow. Whether you're dealing with meeting notes, brainstorming sessions, or voice memos, this chapter will show you how to automate the process, saving time and ensuring nothing falls through the cracks.

This chapter can demonstrate the practical utility of LLMs in day-to-day productivity, adding another layer of automation to your development toolkit.

RAG and Vector

Summary: In this chapter, we delve into the concept of Retrieval-Augmented Generation (RAG) systems, an architectural approach designed to enhance the effectiveness of LLM applications. We'll cover the foundational aspects of RAG, including how to set it up easily within your projects. However, it's important to understand that while RAG can significantly improve LLM performance, it's not an end in itself. We'll discuss the limitations of RAG and how it should be seen as one tool among many in your development toolkit, rather than a complete solution.

This chapter will provide valuable insights into the benefits and limitations of RAG systems, helping readers make informed decisions about when and how to use this approach in their projects.

SiteMap - Feed to RAG

Summary: In this chapter, we explore how to take a website's sitemap and integrate it into a Retrieval-Augmented Generation (RAG) system. We'll discuss the process of chunking the sitemap content to make it searchable, allowing you to interact with it through search, chat, or even by creating a chatbot. By the end of this chapter, you'll have the knowledge to feed customer content into a RAG system, enabling powerful new ways to engage with the material.

This chapter will guide readers through a practical application of RAG, demonstrating how to use it to enhance customer interactions and content management.

Agents to Evaluate the Results

Summary: This chapter focuses on evaluating the accuracy of LLM outputs. We discuss methods for assessing the quality of the results and provide strategies for improving reliability over time.

Standards Checking Tool

Summary: This chapter introduces a standards checking tool that evaluates documents against predefined standards. We cover how to set up the tool to work with any LLM, how to interpret the results, and how to integrate this tool into your content management workflow

Final Thoughts

By now, I've shared with you my experiences and insights on integrating LLMs into your day-to-day solutions. We've covered not only the practical aspects of working with these powerful tools but also the strategies for making them a reliable part of your development workflow.

From setting up local environments to save costs, to testing and mocking with confidence, to abstracting LLMs with data objects—these are skills that will empower you to harness the full potential of LLMs. By adopting these techniques, you'll be able to approach complex problems with a new perspective, equipped with tools that simplify and enhance your work.

As we conclude, it's clear that while the initial hype around LLMs might fade, the practical applications and possibilities they offer are here to stay. Whether you're solving age-old problems or tackling new challenges, the integration of LLMs into your toolkit opens up a world of opportunities to innovate and deliver more effective solutions to your clients.

I hope this book has not only provided you with the technical know-how but also inspired you to explore and push the boundaries of what's possible with LLMs. The skills you've gained will serve you well as you continue to evolve in this ever-changing landscape of technology.

Thank you for joining me on this journey, and I look forward to seeing how you apply these insights to create powerful, intelligent solutions in your work.

This expanded conclusion aims to reinforce the key takeaways from the book while encouraging your readers to apply what they've learned in their own projects.