# API

1PHPD

# 1PHPD – API

**Course Objectives**

By the end of the course, students should:

- Be aware of what is an API
- Be able to create simple JSON API

Time:

- course: 1.5h
- exercises: 2.5h

# *Summary*

1. API
2. PHP JSON

# 1. API

# 1. API

An API (Application Programming Interface) is a set of rules, protocols, and tools for building software and applications.

It specifies how software components should interact and provides a way for different systems to communicate with each other.

APIs are used to enable the integration between different software products and services, making it possible for them to exchange data and functionalities efficiently and securely

# 1. API

APIs play a crucial role in today's software development ecosystem, enabling the rapid development of software applications by reusing existing services and functionalities, thereby saving time, reducing costs, and fostering innovation.

# 1. API

Integration and Interoperability

APIs enable seamless integration and data exchange between different systems and services, regardless of their underlying technology.

You can have APIs build with PHP or Python being part of the same ecosystem and be used by the same frontend application (or other APIs).

# 1. API

Automation

By allowing software to handle tasks automatically, APIs can help streamline operations, reduce the need for manual input, and increase efficiency.

# 1. API

Innovation

APIs provide a foundation for developing new features, services, or products by leveraging existing functionalities, leading to innovation and new business opportunities.

# 1. API

Customization

With APIs, developers can build custom solutions that specifically cater to their needs, enhancing the user experience and adding value to services.

# 1. API

Scalability

APIs facilitate scalability by allowing systems to integrate and share data and functionalities easily, supporting growth and expansion.

# 1. API

Today, all the systems you are using on a daily basis rely and are working thanks to APIs.

Even if APIs are an old concepts, at least since the start of the year 2000, REST API really took of following the reveal of the first IPhone.

Let’s see some example of API that we all use.

# 1. API

PayPal’s API allows e-commerce websites to integrate PayPal’s payment processing capabilities directly into their checkout processes.

This provides a seamless and secure payment experience for users, without the need to leave the merchant’s website to complete a transaction.

# 1. API

The Twitter API enables applications to post tweets, read users' timelines, manage followers, and more.

This can be used by applications to automate content posting, analyze social media engagement, or integrate social media functionalities into their platforms.

# 1. API

Financial market data providers like Bloomberg and Reuters offer APIs that provide real-time financial data, news, and analytics to financial institutions and trading platforms.

These APIs allow for the integration of up-to-date financial information into trading algorithms, risk management systems, and decision-support tools.

# 1. API

Spotify’s Web API allows developers to access Spotify’s music catalog, user data, playlists, and more.

Applications can use this API to create custom playlists, recommend music, or even analyze listening habits.

This API is used each time you are navigating on Spotify, from the website or the mobile application

# 1. API

A REST API (Representational State Transfer Application Programming Interface) is a set of rules and conventions for building web services that allow clients (such as web browsers or mobile applications) to communicate with servers using the HTTP protocol.

REST is an architectural style rather than a strict protocol, and it's designed to take advantage of existing web protocols to make APIs easier to use and more efficient.

# 1. API

A REST API is composed of three main components

- Resources
- Methods
- Representations

# 1. API

Resources

The key abstraction of information in REST and are represented by URLs. A resource can be a document, a picture, a temporal service (e.g., "today's weather in London"), a collection of other resources, etc.

# 1. API

Methods

HTTP methods (also known as verbs) such as GET, POST, PUT, DELETE, and PATCH are used in REST APIs to define the action to be performed on the resources.

For example, GET is used to retrieve a resource, POST is used to create a new resource, and PUT is used to update an existing resource.

# 1. API

Representations

When a client requests a resource, the server sends the client a representation of the resource.

The representation typically includes the resource's data (or the requested part of the resource) along with metadata about the representation. This can be in various formats, such as JSON, XML, HTML, etc.

# 1. API

On a more common world, a REST API is simply a way to get information (or send information) to a server, by sending only the data part and not all the HTML to render the application.

Think about a simple application on your phone, like Uber or Spotify and the equivalent on a browser. No one want to reimplement the same server part two times (or more if you include IPhone).

This is the idea behind REST API, you create one server solution that all frontend (web and mobile) can query.

# 1. API

Most of the time, part of a REST API, you will be using the JSON format, a format that can be understood by all language.

JavaScript Object Notation, more commonly known by the acronym JSON, is an open data interchange format that is both human and machine-readable.

Despite the name JavaScript Object Notation, JSON is independent of any programming language and is a common API output in a wide variety of applications.

# Exercises

# 2. PHP JSON

# 2. PHP JSON

We saw in a previous chapter that we can define most of the method as part of a CRUD operations.

| Operation | HTTP | SQL |
|---|---|---|
| Create | POST | INSERT INTO |
| Read | GET | SELECT |
| Update | PUT/PATCH | UPDATE |
| Delete | DELETE | DELETE |

# 2. PHP JSON

This would be the main methods used when creating a REST API, allowing interactions on all resources with the previous value.

- Create as a way to add and insert new values
- Read to get and display some data
- Update to change an existing value
- Delete to remove an item from a database

# 2. PHP JSON

First, we need to work on getting the information related to the request that is being performed.

- Which URL is being called?
- What is the method used?
- Is there some content being sent?
  - Is it in the URL?
  - Is there a body?
  - Is there some specific headers?

# 2. PHP JSON

URL Parsing

First, we need to know what URL is currently being called. This will help us know if we are working with “users” item (url would be “/users”) or some “messages” (url would be “/messages”).

For that we can use the superglobal from PHP storing the URL information “$_SERVER” and then parse the result to only keep the value after the domain name (or after localhost if working locally)

# 2. PHP JSON

```php
// Parse the request
$uri = parse_url($_SERVER['REQUEST_URI'], PHP_URL_PATH);
$uri = explode('/', $uri);

// Verify the request URI
if ($uri[1] !== 'users') {
    http_response_code(404);
    echo json_encode(['error' => 'Not Found']);
    exit;
}
```

# 2. PHP JSON

In the previous example, we display and return an error if the URL is not the “users”.

In a normal case, when working with an API with multiple endpoints we would use something a bit more smarter, maybe a “switch” statement or a list of “if/else” to allow multiple values.

# 2. PHP JSON

We also remark two new method that we did not use before

- “http_response_code”: Get or Set the HTTP response code

- “json_encode”: Returns a string containing the JSON representation of the supplied value.
  - If the parameter is an array or object, it will be serialized recursively.
  - JSON is a simple and lightweight way to represent data, really close to a JavaScript Object (but not exactly the same)

# 2. PHP JSON

Example from a JSON API

This is the result of the REST API from GitHub with a specific user name (here using “sheplu”)

Link


```
{
  "login": "sheplu",
  "id": 1133916,
  "node_id": "MDQ6VXNlcjExMzM5MTY=",
  "avatar_url": "https://avatars.githubusercontent.com/u/1133916?v=4",
  "gravatar_id": "",
  "url": "https://api.github.com/users/sheplu",
  "html_url": "https://github.com/sheplu",
  "followers_url": "https://api.github.com/users/sheplu/followers",
  "following_url": "https://api.github.com/users/sheplu/following{/other_user}",
  "gists_url": "https://api.github.com/users/sheplu/gists{/gist_id}",
  "starred_url": "https://api.github.com/users/sheplu/starred{/owner}{/repo}",
  "subscriptions_url": "https://api.github.com/users/sheplu/subscriptions",
  "organizations_url": "https://api.github.com/users/sheplu/orgs",
  "repos_url": "https://api.github.com/users/sheplu/repos",
  "events_url": "https://api.github.com/users/sheplu/events{/privacy}",
  "received_events_url": "https://api.github.com/users/sheplu/received_events",
  "type": "User",
  "site_admin": false,
  "name": "Jean Burellier",
  "company": "Oromys",
  "blog": "",
  "location": "Lyon, France",
  "email": null,
  "hireable": true,
  "bio": null,
  "twitter_username": "shepsheplu",
  "public_repos": 135,
  "public_gists": 3,
  "followers": 83,
  "following": 13,
  "created_at": "2011-10-17T18:17:25Z",
  "updated_at": "2024-02-26T07:48:23Z"
}
```

# 2. PHP JSON

An HTTP code, also known as an HTTP status code, is a standardized numerical code that is sent from a server to a client to indicate the outcome of the client's request.

These codes are part of the HTTP (Hypertext Transfer Protocol) response message that accompanies the data being retrieved by the request.

HTTP status codes are divided into five categories, each representing a specific type of response

# 2. PHP JSON

1xx - Informational

These status codes indicate a provisional response and signify that the request has been received and the process is continuing.

2xx - Success

These codes indicate that the client's request was successfully received, understood, and accepted.

# 2. PHP JSON

3xx - Redirection

These codes indicate that further action needs to be taken by the client in order to complete the request.

4xx - Client Error

These codes indicate an error that the client made, such as a bad request or a request for a resource that does not exist.

# 2. PHP JSON

5xx - Server Error

These codes indicate that the server failed to fulfill an apparently valid request made by the client.

# 2. PHP JSON

After getting the URL from the request, we need to check the method. Again we can rely on the PHP superglobal “$_SERVER”.

This should return the value of the request you used. By default a request done by a browser would be a “GET”

```php
$requestMethod = $_SERVER["REQUEST_METHOD"];
```

# 2. PHP JSON

With a simple switch statement, we can then return different value based on the method used to do the request.

```php
// Handle the request based on the method
switch ($requestMethod) {
    case 'GET':
        // Handle GET request
        // This is where you would retrieve and send data from the database
        echo json_encode(['method' => 'GET', 'message' => 'Handle GET request']);
        break;
    case 'POST':
        // Handle POST request
        // This is where you would handle data creation
        echo json_encode(['method' => 'POST', 'message' => 'Handle POST request']);
        break;
    case 'PUT':
```

# 2. PHP JSON

This simple example is just to display the core methods that are necessary to create a simple REST API, with hardcoded values in place of a more normal response.

On a true API, you would get some data from the request - like you already know using for example “$_POST” - and using that set of data to do a request to a database.

# 2. PHP JSON

On an normal use case for a REST API, you would have the following steps

- Check which URL and method are used
- Sanitize and validate user inputs
- Call the good method (for example “getUser” if the URL is “/users” and the method is “GET”)
- Process all information, doing a call to a database (or another API, or ready a file)
- Return a JSON Object or an Error (formatted as a JSON Object)

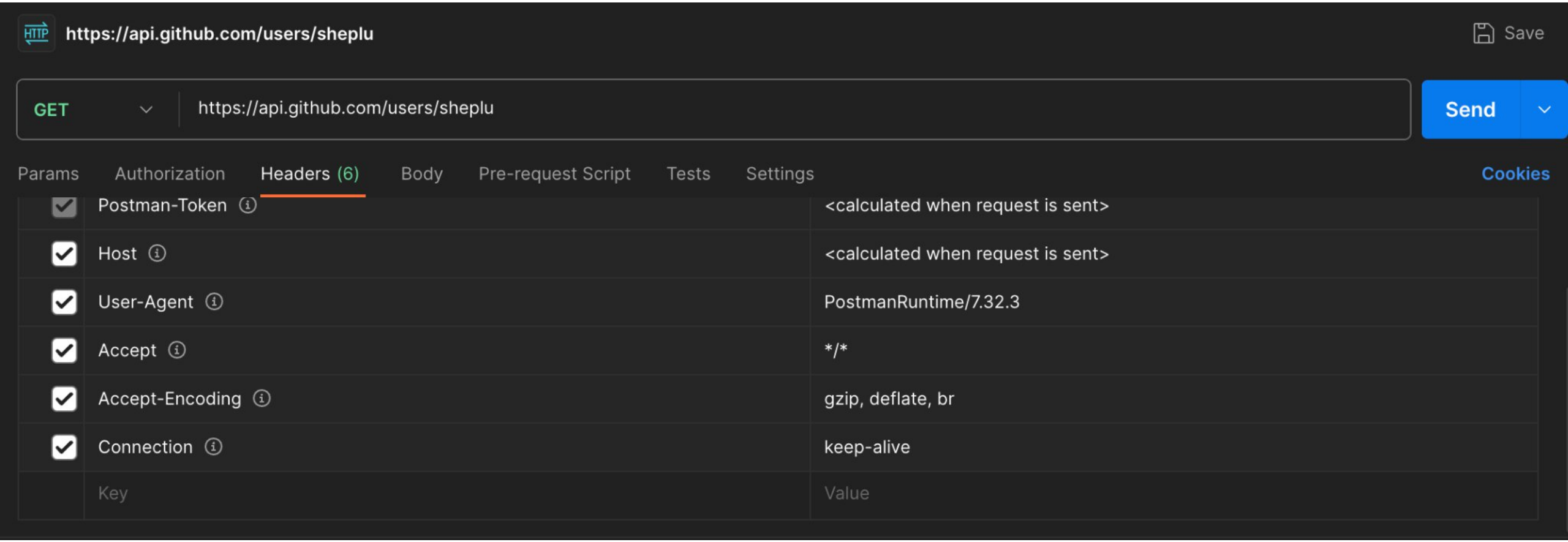
# 2. PHP JSON

To test and try your API, you can use tools like Postman or cURL. The first one provides a simple graphical interface while the second can only be used through a command line.

Using one of the two tools, you will be able to test your API but also other public APIs (like the one on GitHub).

You should be able to see all the information we talked to during this API presentation, including HTTP Status Code, but also Headers and Body.

# 2. PHP JSON


https://api.github.com/users/sheplu
Save
GET
https://api.github.com/users/sheplu
Send
Params
Authorization
Headers (6)
Body
Pre-request Script
Tests
Settings
Cookies
Postman-Token
<calculated when request is sent>
Host
<calculated when request is sent>
User-Agent
PostmanRuntime/7.32.3
Accept
*/*
Accept-Encoding
gzip, deflate, br
Connection
keep-alive
Key
Value

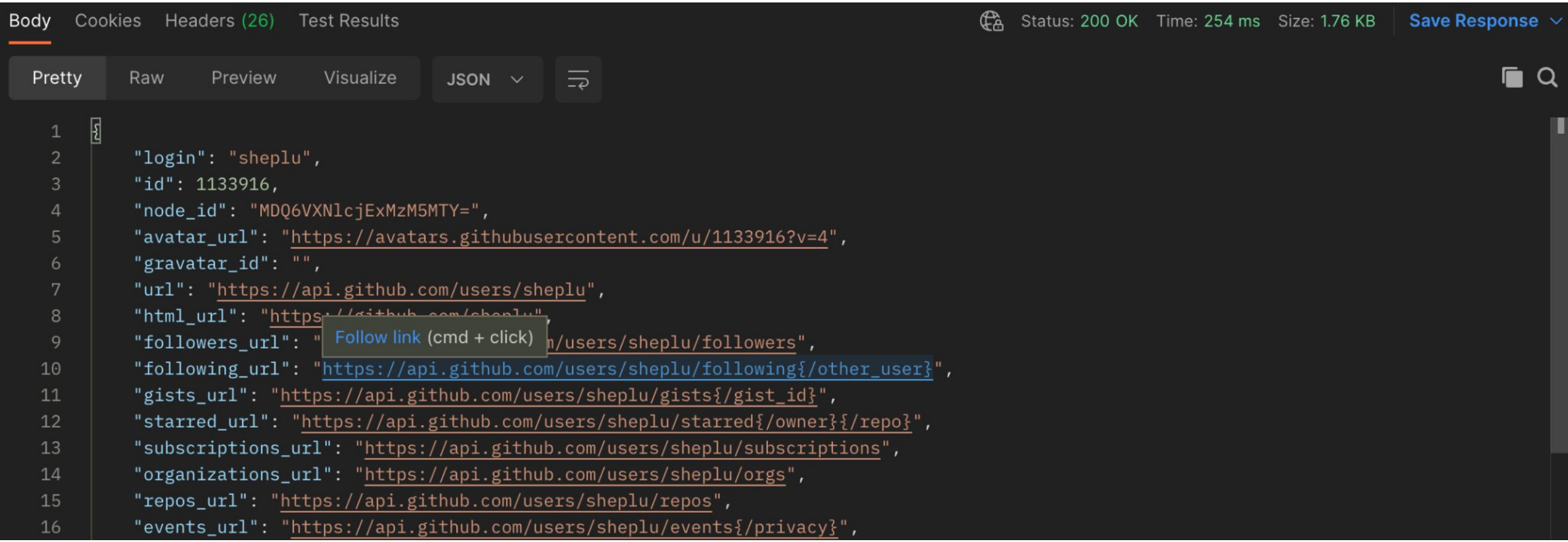
# 2. PHP JSON

In the previous slide we can see the part where - on Postman - you configure and create your request.

Here we define our request as a “GET” request, and we set the URL to query the GitHub users with the value “sheplu”

We can also see some headers that are sent, and are configured by default by Postman. You can change them or add new one if needed

# 2. PHP JSON

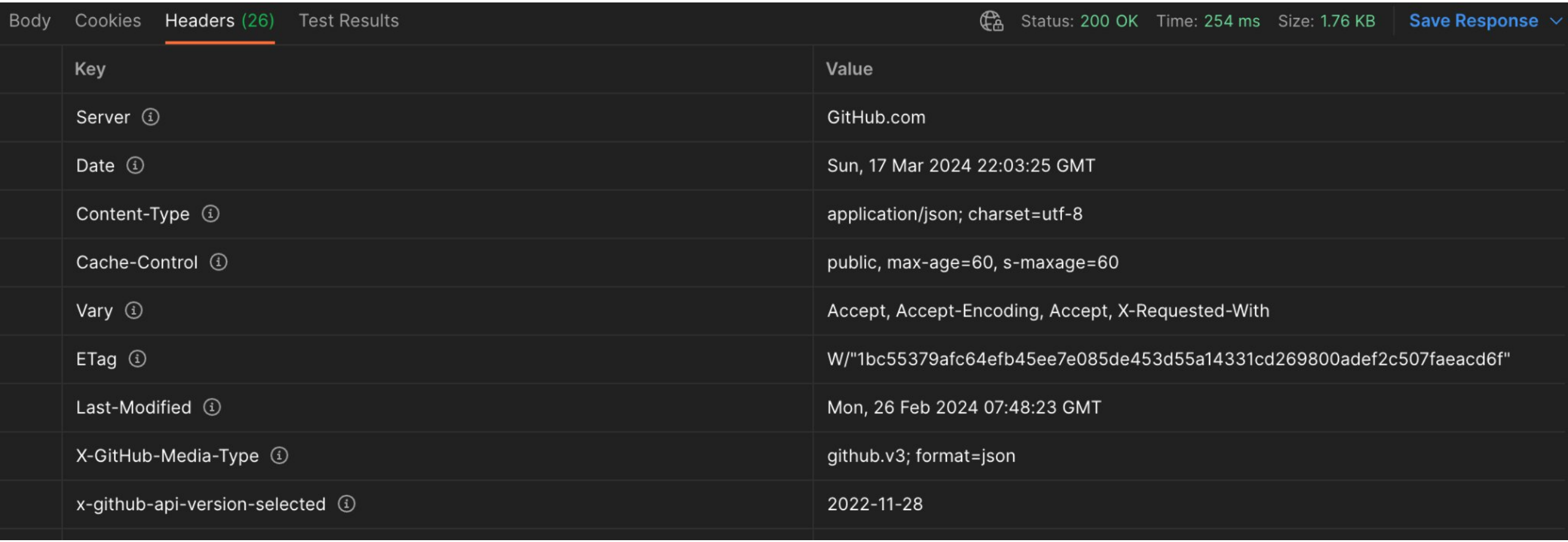

Body
Cookies
Headers (26)
Test Results
Status: 200 OK
Time: 254 ms
Size: 1.76 KB
Save Response
Pretty
Raw
Preview
Visualize
JSON
Follow link (cmd + click)
{
"login": "sheplu",
"id": 1133916,
"node_id": "MDQ6VXNlcjExMzM5MTY=",
"avatar_url": "https://avatars.githubusercontent.com/u/1133916?v=4",
"gravatar_id": "",
"url": "https://api.github.com/users/sheplu",
"html_url": "https
"followers_url":
/users/sheplu/followers",
"following_url": "https://api.github.com/users/sheplu/following{/other_user}",
"gists_url": "https://api.github.com/users/sheplu/gists{/gist_id}",
"starred_url": "https://api.github.com/users/sheplu/starred{/owner}{/repo}",
"subscriptions_url": "https://api.github.com/users/sheplu/subscriptions",
"organizations_url": "https://api.github.com/users/sheplu/orgs",
"repos_url": "https://api.github.com/users/sheplu/repos",
"events_url": "https://api.github.com/users/sheplu/events{/privacy}",

# 2. PHP JSON

The first part of the response is comprise of all information we mentioned before.

- The HTTP Status code: 200
- A body with some data: a JSON Object
- The size of the data: 1.76Kb
- The time the request took: 254ms
  - This value is computed by Postman

# 2. PHP JSON



Body Cookies Headers (26) Test Results Status: 200 OK Time: 254 ms Size: 1.76 KB Save Response

| Key | Value |
| --- | --- |
| Server | GitHub.com |
| Date | Sun, 17 Mar 2024 22:03:25 GMT |
| Content-Type | application/json; charset=utf-8 |
| Cache-Control | public, max-age=60, s-maxage=60 |
| Vary | Accept, Accept-Encoding, Accept, X-Requested-With |
| ETag | W/"1bc55379afc64efb45ee7e085de453d55a14331cd269800adef2c507faeacd6f" |
| Last-Modified | Mon, 26 Feb 2024 07:48:23 GMT |
| X-GitHub-Media-Type | github.v3; format=json |
| x-github-api-version-selected | 2022-11-28 |

# 2. PHP JSON

If we click on the “headers” tabs, we can then see all the headers that are part of the response from the servers at GitHub.

Some information are really important, like the “Content-Type” that is used to signal to your browser the type of data that is being send.

In our example we can see that some “application/json” with a “charset=utf-8” is being returned.

# Exercises