Resumable file upload in PHP: Handle large file uploads in an elegant way

hackernoon.com/resumable-file-upload-in-php-handle-large-file-uploads-in-an-elegant-way-e6c6dfdeaedb

> Ever struggled with large file upload in PHP? Wondered if you could continue uploading where you left off without re-uploading whole data again in case of any interruptions? If this sounds familiar to you, then keep reading.

Photo by rawpixel.com on Unsplash

File upload is a common task we do in almost all of our modern web projects. With all different tools available, it is not that hard to implement file upload feature in any language. But, still, when it comes to large file upload things gets a bit complicated.

Say, you are trying to upload a fairly large file. You have been waiting for more than an hour already and the upload is at 90%. Then suddenly, your connection drops or browser crashed. The upload is aborted and you need to start from the beginning. Frustrating, isn’t it? Even worse, if you are in a slow connection, like many places in the world, no matter how often you try you will only be able to upload first part of the upload every time.

In this post, we will see an attempt to solve this problem in PHP by uploading files in resumable chunks using tus protocol.

Resumable File Upload in PHP — Demo

What is tus?

Tus is a HTTP based open protocol for resumable file uploads. Resumable means we can carry on where we left off without re-uploading whole data again in case of any interruptions. An interruption may happen willingly if the user wants to pause, or by accident in case of a network issue or server outage.

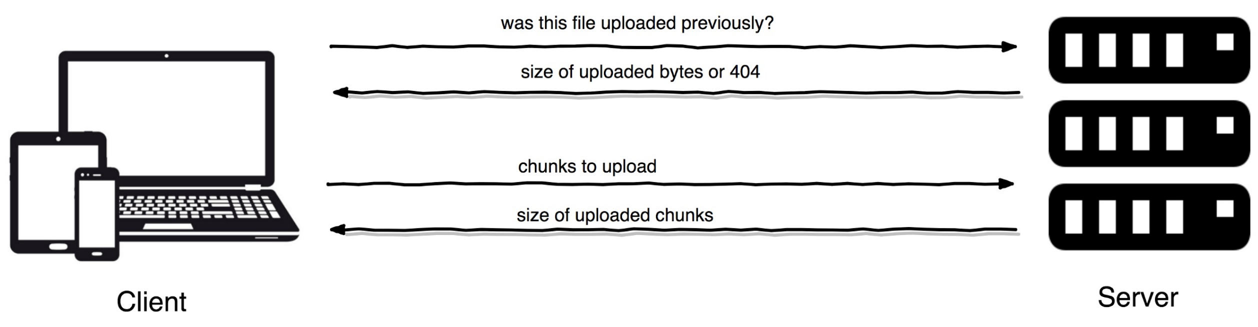
> Tus protocol was adopted by Vimeo in May 2017.

Why tus?

Quoting from Vimeo’s blog:

> We decided to use tus in our upload stack because the tus protocol standardizes the process of uploading files in a concise and open manner. This standardization will allow API developers to focus more on their application-specific code, and less on the upload process itself.

Another main benefit of uploading a file this way is that you can start uploading from a laptop and even continue uploading the same file from your mobile or any other device. This is a great way to enhance your user experience.


was this file uploaded previously?
size of uploaded bytes or 404
chunks to upload
size of uploaded chunks
Client
Server

Pic: Basic Tus Architecture

Getting Started

Let's start by adding our dependency.

```
$ composer require ankitpokhrel/tus-php
```

tus-php is a framework agnostic pure PHP server and client implementation for the tus resumable upload protocol v1.0.0.

ankitpokhrel/tus-php
*tus-php - 🚀A pure PHP server and client for the tus resumable upload protocol v1.0.0*github.com

> **Update**: Vimeo is now using TusPHP in v3 of their Official PHP library for the Vimeo API.

Creating a server to handle our requests

This is how a simple server looks like.

```
// server.php

$server   = new \TusPhp\Tus\Server('redis');
$response = $server->serve();

$response->send();

exit(0); // Exit from current PHP process.
```

You need to configure your server to respond to a specific endpoint. For example, in Nginx, you would do something like this:

```
# nginx.conf

location /files {
    try_files $uri $uri/ /path/to/server.php?$query_string;
}
```

Let's assume that the URL to our server is *http://server.tus.local.*So, based on above nginx configuration we can access our tus endpoints using *http://server.tus.local/files.*

Now, we have following RESTful endpoints available to work with.

```
# Gather information about server's current configuration
OPTIONS /files

# Check if the given upload is valid
HEAD /files/{upload-key}

# Create a new upload
POST /files

# Resume upload created with POST
PATCH /files/{upload-key}

# Delete the previous upload
DELETE /files/{upload-key}
```

Check out the protocol details for more info about the endpoints.

> If you are using any frameworks like Laravel, instead of modifying your server config, you can define routes to all tus based endpoints in your framework route file. We will cover this in detail in another tutorial.

Handling upload using tus-php client

Once the server is in place, the client can be used to upload a file in chunks. Let us start by creating a simple HTML form to get input from the user.

```
<form action="upload.php" method="post" enctype="multipart/form-data">
    <input type="file" name="tus_file" id="tus-file" />
    <input type="submit" value="Upload" />
</form>
```

After a form is submitted, we need to follow few steps to handle the upload.

1. **Create a tus-php client object**

```
// Tus client

$client = new \TusPhp\Tus\Client('http://server.tus.local');
```

The first parameter in the above code is your tus server endpoint.

2. **Initialize client with file metadata**

To keep an upload unique, we need to use some identifier to recognize the upload in upcoming requests. For this, we will have to generate a unique upload key which can be used to resume the upload later. You can either explicitly provide an upload key or let the system generate a key by itself.

```
// Set upload key and file meta

$client->setKey($uploadKey)
    ->file($_FILES['tus_file']['tmp_name'], 'your file name');
```

If you don't provide upload key explicitly, above step will be something like this:

```
$client->file($_FILES['tus_file']['tmp_name'], 'your file name');

$uploadKey = $client->getKey(); // Unique upload key
```

3. **Upload a chunk**

```
// $chunkSize is size in bytes, i.e. 5000000 for 5 MB

$bytesUploaded = $client->upload($chunkSize);
```

Next time, when you want to upload another chunk you can use same upload key to continue.

```
// To resume upload in next request

$bytesUploaded = $client->setKey($uploadKey)->upload($chunkSize);
```

Once the upload is complete, the server verifies upload against the checksum to make sure the uploaded file is not corrupt. The server will use `sha256` algorithm by default to verify the upload.

> The full implementation of the demo video above can be found here.

Handling upload using tus-js-client

Uppy is a sleek, modular file uploader plugin developed by same folks behind tus protocol. You can use uppy to seamlessly integrate official tus-js-client with a *tus-php* server. That means we are using php implementation of a server and js implementation of a client.

```
uppy.use(Tus, {
  endpoint: 'https://server.tus.local/files/', // your tus endpoint
  resume: true,
  autoRetry: true,
  retryDelays: [0, 1000, 3000, 5000]
})
```

> Check out more details in uppy docs and example implementation here.

Partial Uploads

The *tus-php* Server supports concatenation extension and is capable of concatenating multiple uploads into a single one enabling clients to perform parallel uploads and to upload non-contiguous chunks.

Partial file upload using tus-php

The full example of partial uploads can be found here.

Final Words

The tus-php project itself is still in its initial stage. Some sections might change in the future. Three different implementation examples can be found in the example folder. Feel free to try and report any issues found. Pull requests and project recommendations are more than welcome.

Happy Coding!