Sure! Here's a step-by-step guide on how to implement the process you described in a Laravel Lumen application:

1.) Create a new route in your routes/web.php file that will handle the file upload. You can use the post method and specify the URL where the file will be uploaded, for example:

```
$router->post('/upload', 'UploadController@upload');
```

2.) In your UploadController, define the upload method to handle the file upload. This method should receive the file in the POST request and return an ID to the client. You can use the store method provided by Laravel to store the uploaded file and generate a unique ID for it. Here's an example:

```
public function upload(Request $request)

{

    $file = $request->file('file');

    $id = uniqid();

    $path = $file->storeAs('uploads', $id);



    return response()->json(['id' => $id]);

}
```

This code assumes that the uploaded file is sent in a field called file. The storeAs method will store the file in the uploads folder and use the ID as the filename.

3.) In your client-side code, send a POST request to the /upload endpoint with the file attached as multipart/form-data. The server will return an ID that you can use to associate the metadata with the file. Here's an example using JavaScript and the Fetch API:

```
const fileInput = document.querySelector('#file-input');

const formData = new FormData();

formData.append('file', fileInput.files[0]);



fetch('/upload', {

  method: 'POST',

  body: formData
```

```
})
.then(response => response.json())
.then(data => {
  const fileId = data.id;
  // Use the fileId to send the metadata to the server
});
```

4.) Once the client receives the ID, it can send the metadata to the server along with the ID. You can use another endpoint to handle the metadata, for example:

```
$router->post('/metadata', 'MetadataController@save');
```

5.) In your MetadataController, define the save method to handle the metadata. This method should receive the ID and the metadata and re-associate them with the file. You can retrieve the file from the storage using the ID and update its metadata. Here's an example:

```
public function save(Request $request)
{
    $id = $request->input('id');
    $metadata = $request->input('metadata');

    $path = Storage::path("uploads/$id");
    $file = new SplFileInfo($path);
    $file->metadata = $metadata;

    return response()->json(['success' => true]);
}
```

This code assumes that the metadata is sent in a field called metadata. The SplFileInfo class is used to retrieve the file from the storage and update its metadata.

That's it! You now have a system where the client can upload a file first and then send the

metadata separately, and the server can re-associate them using a unique ID.