

It really became clear that I didn’t know what I was doing. So like any good developer, I looked to copy. So I took a look at the public API’s of the three social media giants.

**Twitter**

API Doc : <https://developer.twitter.com/en/docs/media/upload-media/api-reference/post-media-upload>

Twitter has two different ways to upload files. The first is sort of a “chunked” way, which I assume is because you can upload some pretty large videos these days. And a more simple way for just uploading general images, let’s focus on the latter.

It’s a multi-part form, but returns JSON. Boo.

The very very interesting part about the API however, is that it allows uploading the actual data in two ways. Either you can upload the raw binary data as you typically would in a multipart form post, or you could actually serialise the file as a Base64 encoded string, and send that as a parameter.

Base64 encoding a file was interesting to me because theoretically (And we we will see later, definitely), we can send this string data any way we like. I would say that of all the C# SDKs I looked at, I couldn’t find any actually using this Base64 method, so there weren’t any great examples to go off.

Another interesting point about this API is that you are **uploading “media**”, and then at a later date attaching that to an actual object (For example a tweet). So if you wanted to tweet out an image, it seems like you would (correct me if I’m wrong) upload an image, get the ID returned, and then create a tweet object that references that media ID. For my use case, I certainly didn’t want to do a two step process like this.

#### LinkedIn

API Doc : <https://developer.linkedin.com/docs/guide/v2/shares/rich-media-shares#upload>

LinkedIn was interesting because it’s a pure JSON API. All data POSTs contain JSON payloads, similar to the API I was creating. Wouldn’t you guess it, they use a multipart form data too!

Similar to Twitter, they also have this concept of uploading the file first, and attaching it to where you actually want it to end up second. And I totally get that, it’s just not what I want to do.

#### Facebook

API Doc : <https://developers.facebook.com/docs/graph-api/photo-uploads>

Facebook uses a Graph API. So while I wanted to take a look at how they did things, so much of their API is not really relevant in a RESTful world. They do use multi-part forms to upload data, but it’s kinda hard to say how or why that is the case,. Also at this point, I couldn’t get my mind off how Twitter did things!

### **So Where Does That Leave Us?**

Well, in a weird way I think I got what I expected, That multipart forms were well and truly alive. It didn’t seem like there was any great innovation in this area. In some cases, **the use of multipart forms didn’t look so brutal because they didn’t need to upload metadata at the same time**. Therefore simply sending a file with no attached data didn’t look so out of place in a JSON API. However, I did want to send metadata in the same payload as the image, not have it as a two step process.

Twitter’s use of Base64 encoding intrigued me. It seemed like a pretty good option for sending data across the wire irrespective of how you were formatting the payload. You could send a Base64 string as JSON, XML or Form Data and it would all be handled the same. It’s definitely proof of concept time!

### **Base64 JSON API POC**

What we want to do is just test that we can **upload images as a Base64 string**, and we don’t have any major issues within a super simple scenario. Note that these examples are in C# .NET Core, but again, if you are using any other language it should be fairly simple to translate these.

First, we need our upload JSON Model. In C# it would be :

public class UploadCustomerImageModel

{

public string Description { get; set; }

public string ImageData { get; set; }

}

Not a whole lot to it. Just a description field that can be freetext for a user to describe the image they are upload, and an imagedata field that will hold our Base64 string.

For our controller :

[HttpPost("{customerId}/images")]

public FileContentResult UploadCustomerImage(int customerId, [FromBody] UploadCustomerImageModel model)

{

//Depending on if you want the byte array or a memory stream, you can use the below.

var imageDataByteArray = Convert.FromBase64String(model.ImageData);

//When creating a stream, you need to reset the position, without it you will see that you always write files with a 0 byte length.

var imageDataStream = new MemoryStream(imageDataByteArray);

imageDataStream.Position = 0;

//Go and do something with the actual data.

//\_customerImageService.Upload([...])

//For the purpose of the demo, we return a file so we can ensure it was uploaded correctly.

//But otherwise you can just return a 204 etc.

return File(imageDataByteArray, "image/png");

}

Again, fairly damn simple. We take in the model, then C# has a great way to convert that string into a byte array, or to read it into a memory stream. Also note that as we are just building a proof of concept, I echo out the image data to make sure that it’s been received, read, and output like I expect it would, but not a whole lot else.

Now let’s open up postman, our JSON payload is going to look a bit like :

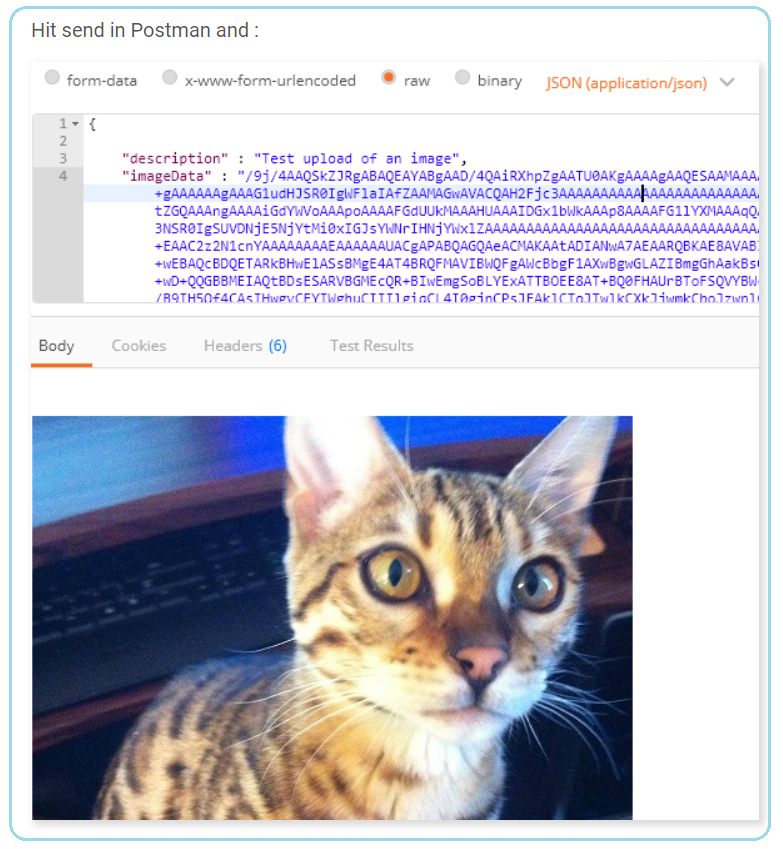
{

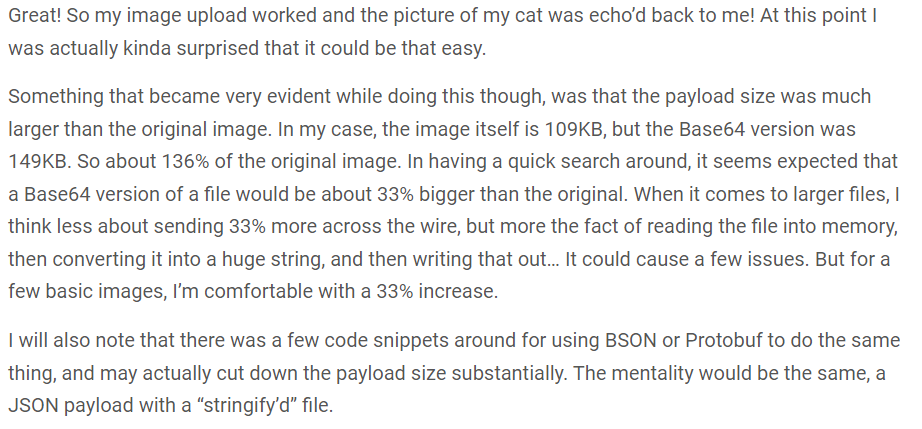
"description" : "Test upload of an image",

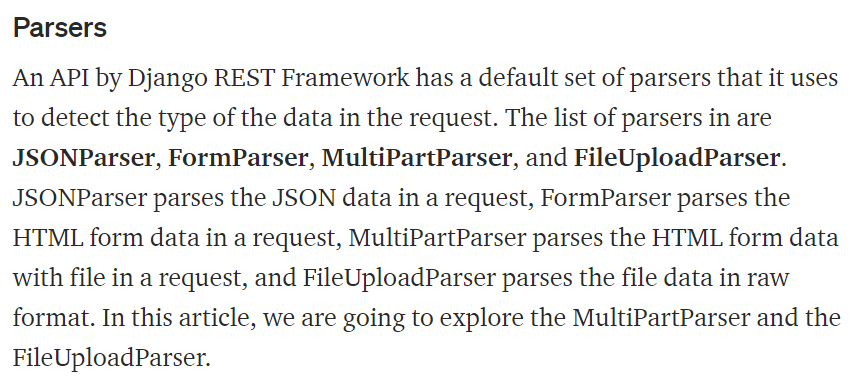
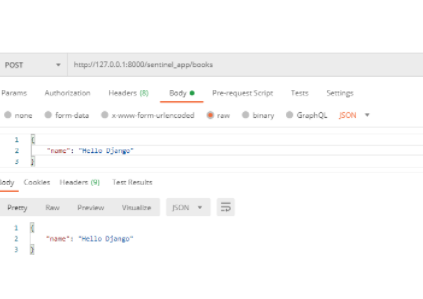
"imageData" : "/9j[...]"

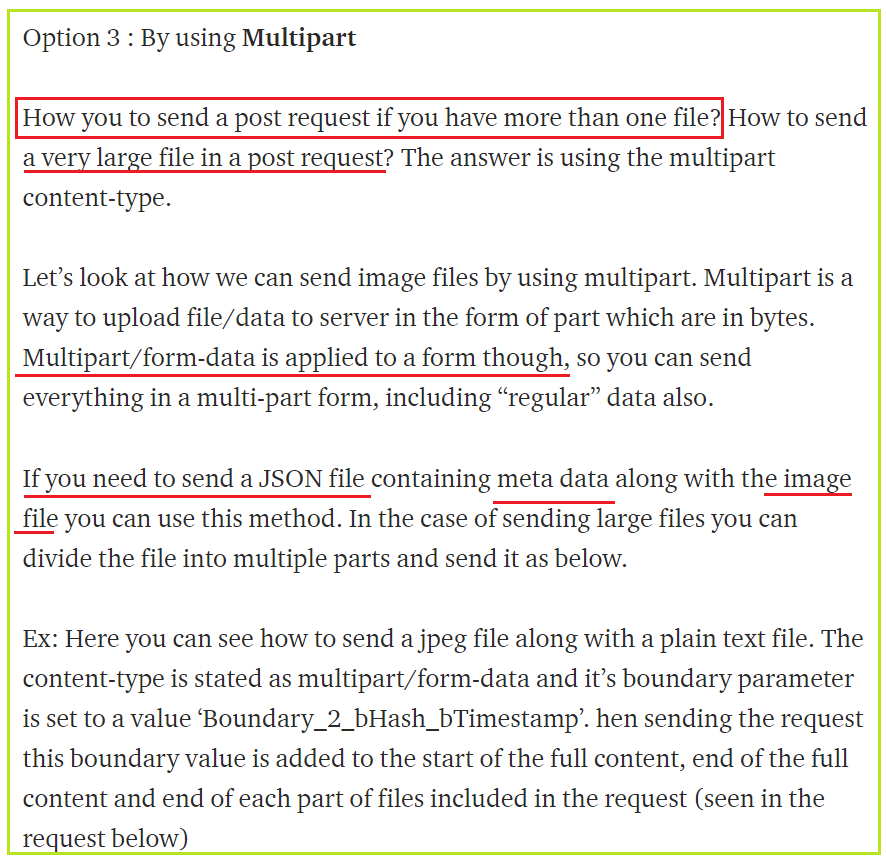
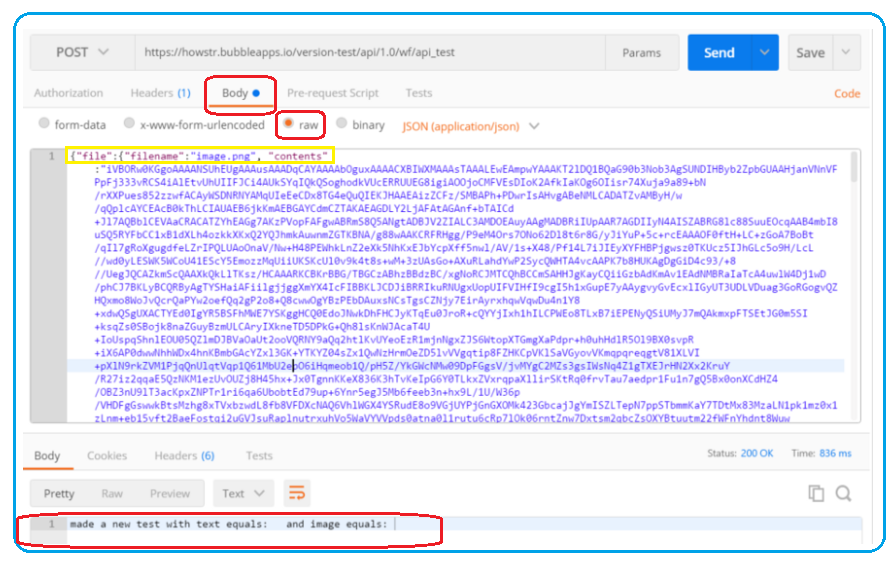
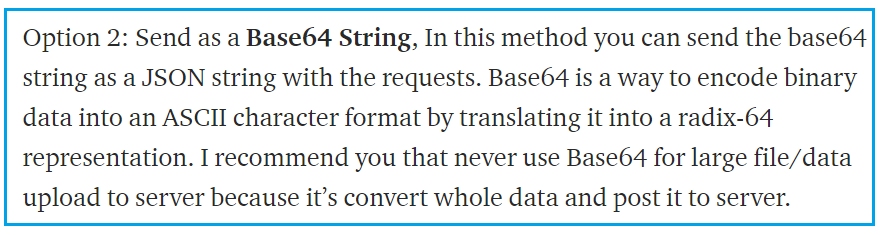
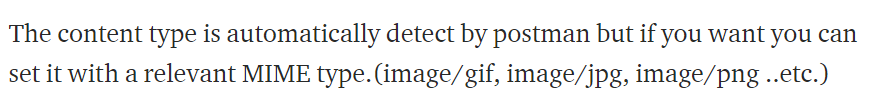
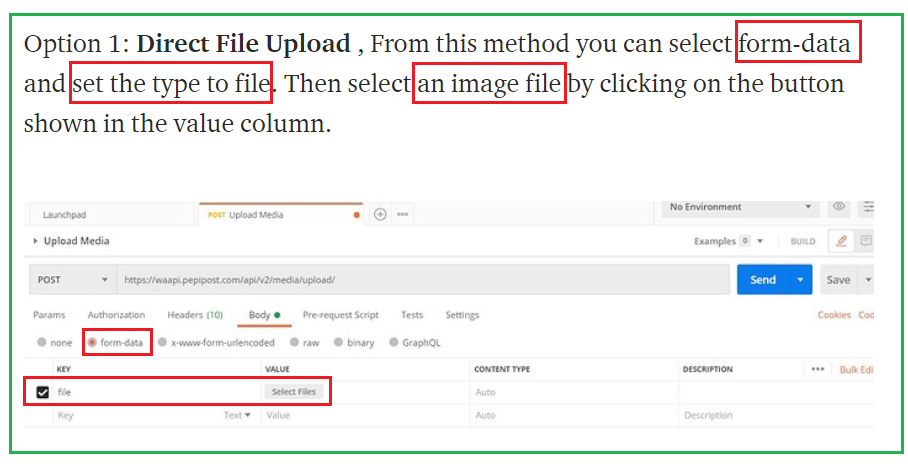
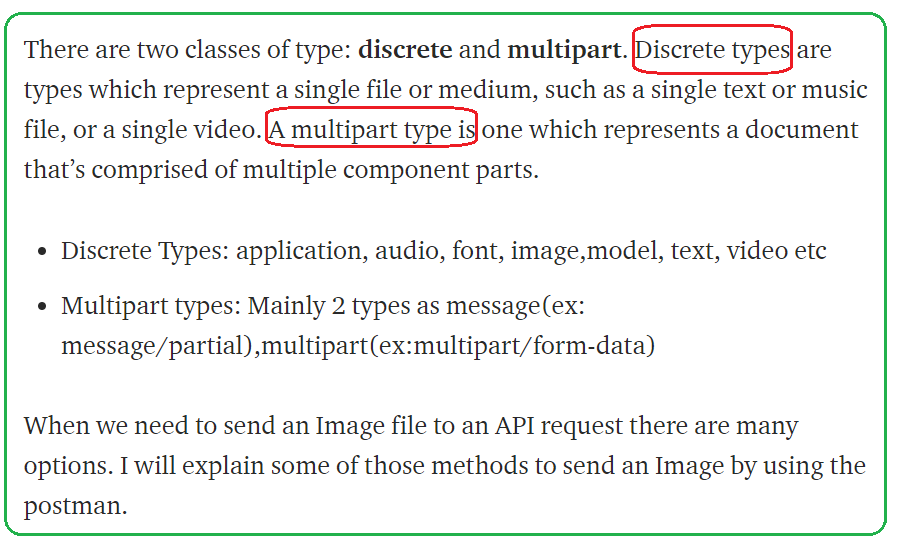
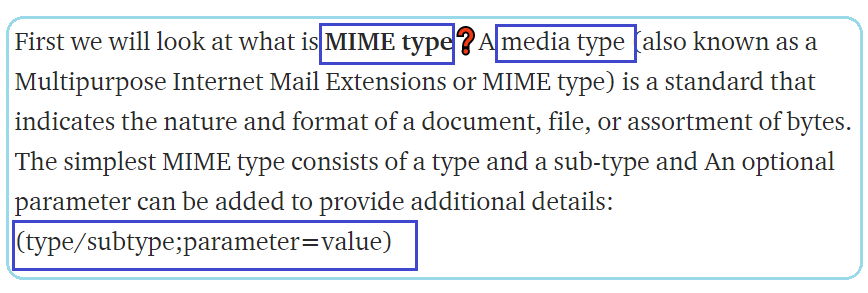
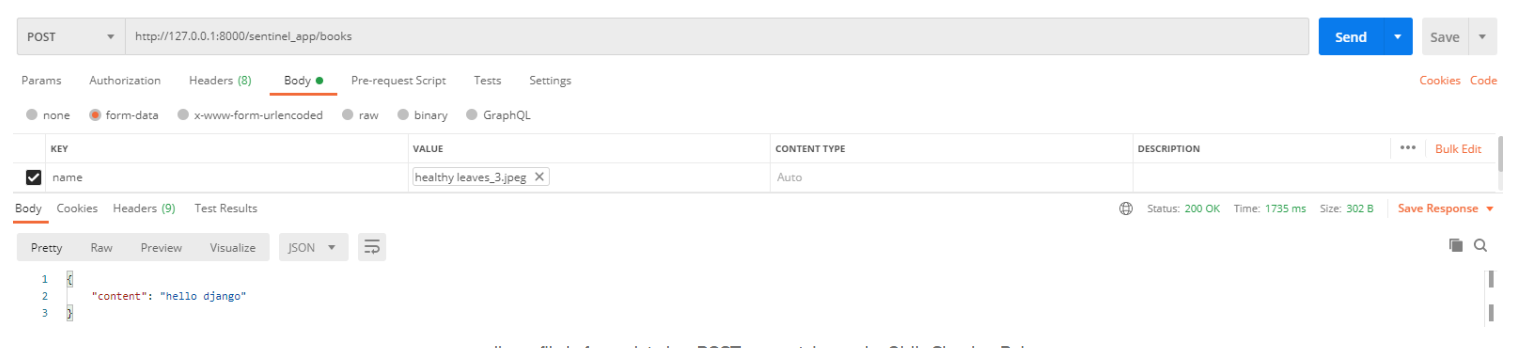
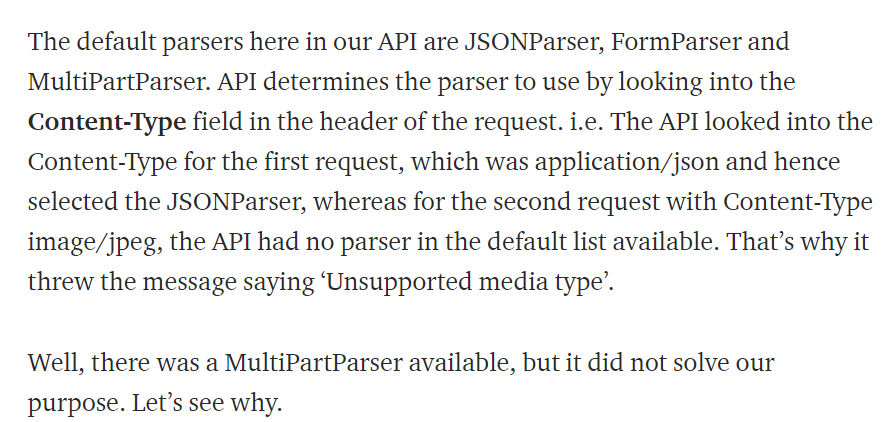
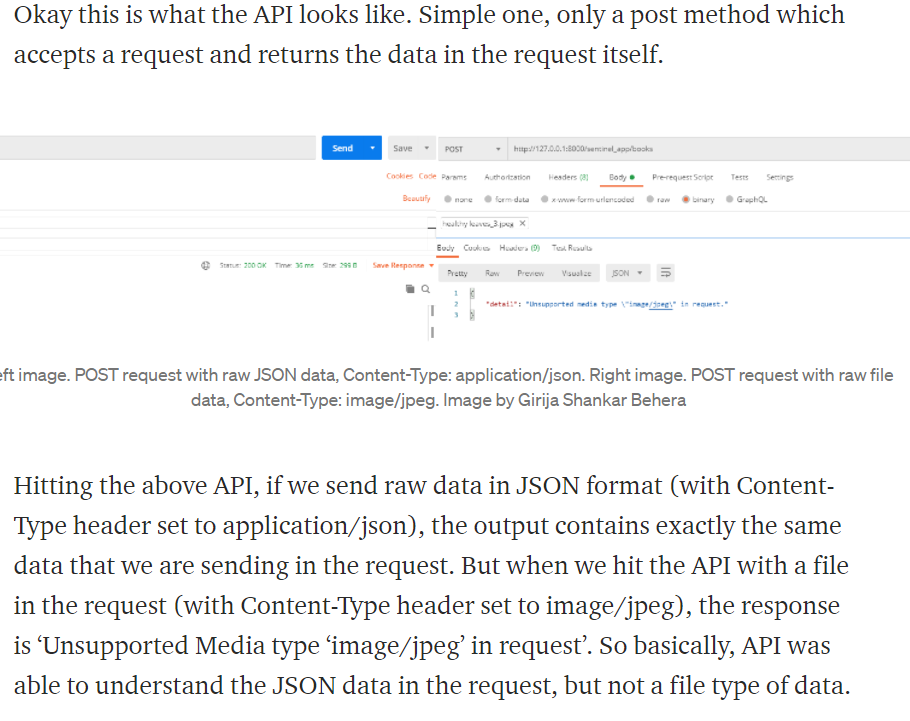
}

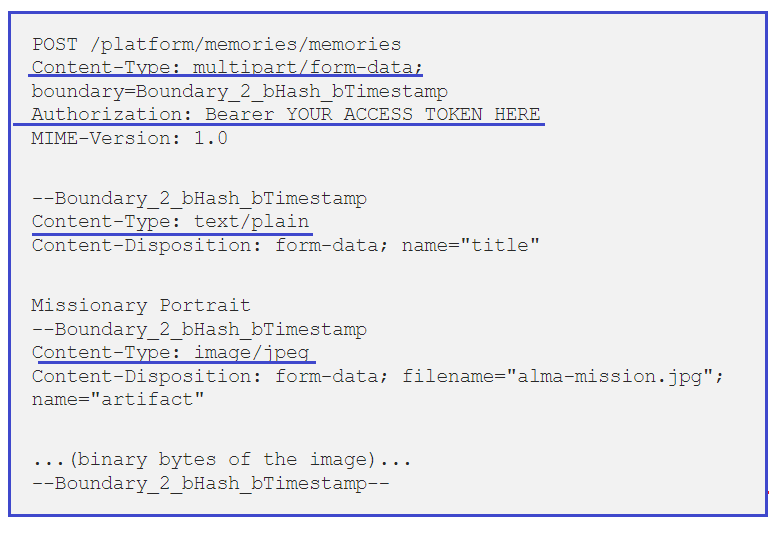
I’ve obviously truncated imagedata down here, but a super simple tool to turn an image into a Base64 is something like [this website here](https://www.base64-image.de/). I would also note that when you send your payload, it should be without the data:image/jpeg;base64, prefix that you sometimes see with online tools that convert images to strings.

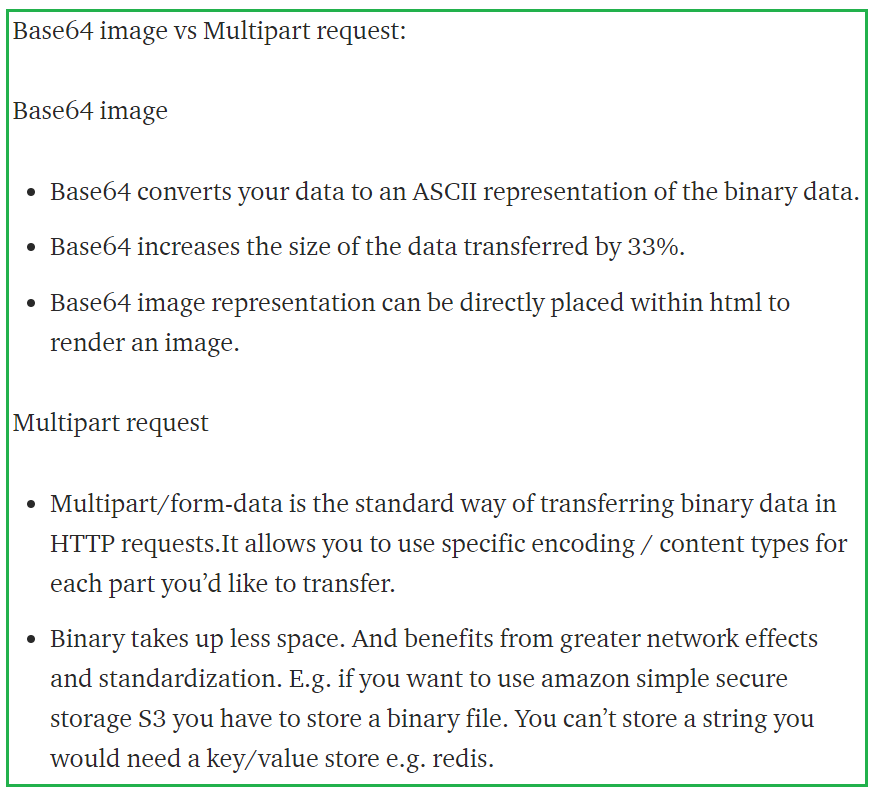
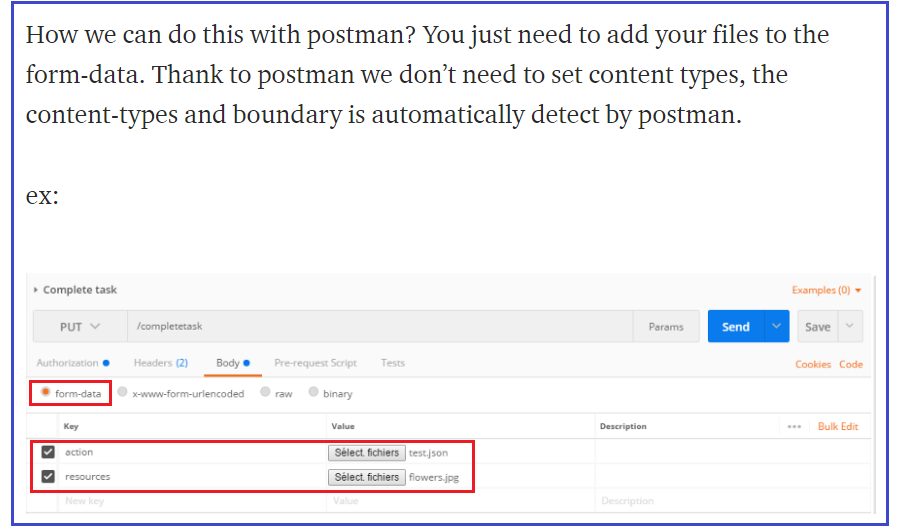


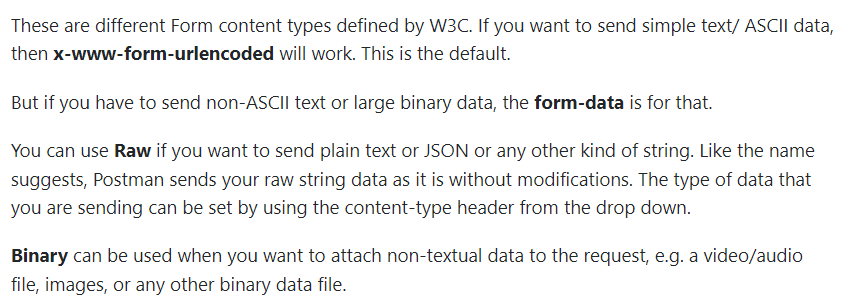


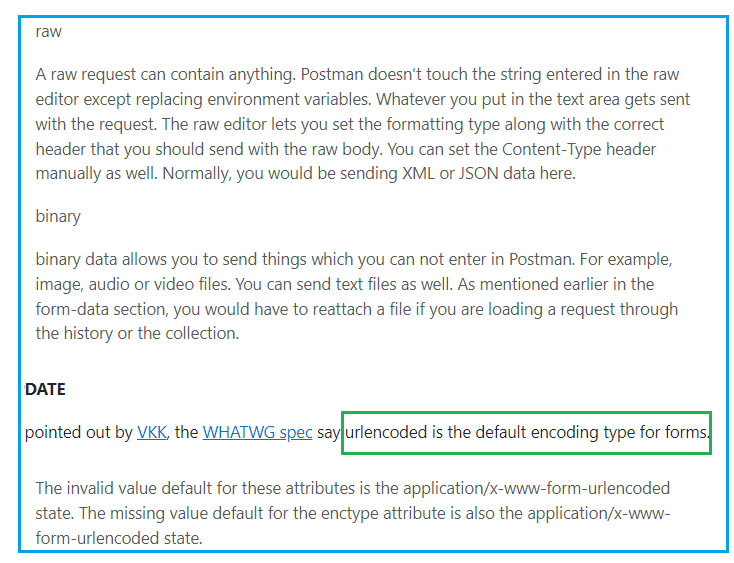
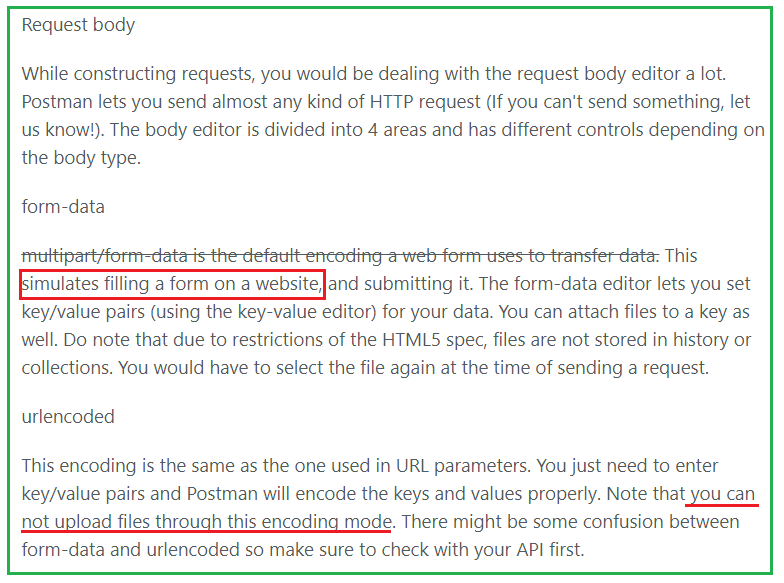
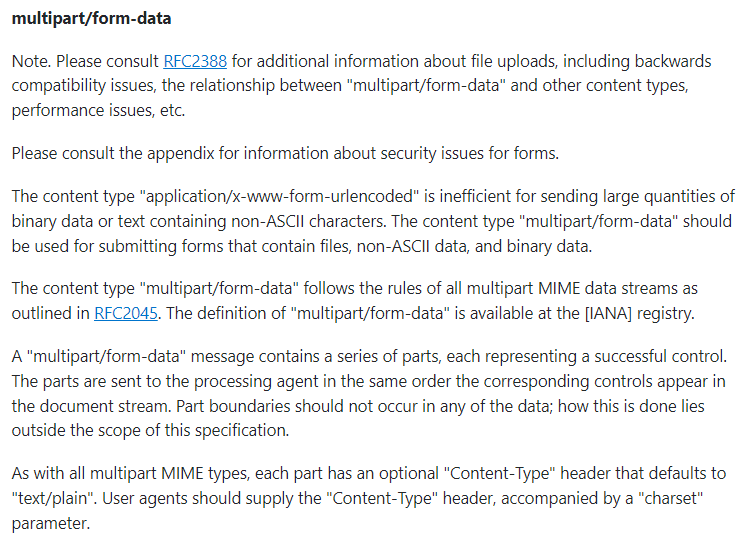
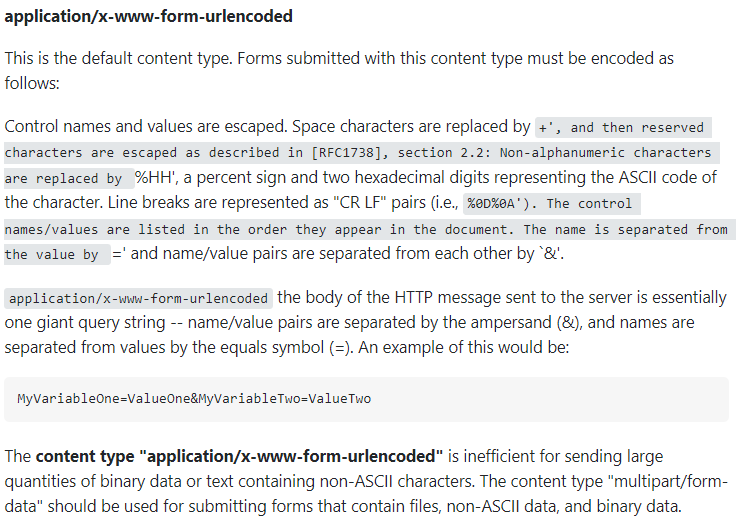










[What are the differences between application/json and application/x-www-form-urlencoded?](https://stackoverflow.com/questions/9870523/what-are-the-differences-between-application-json-and-application-x-www-form-url)

request.ContentType = "application/json; charset=utf-8";

and

request.ContentType = "application/x-www-form-urlencoded";

[json](https://stackoverflow.com/questions/tagged/json)[post](https://stackoverflow.com/questions/tagged/post)[content-type](https://stackoverflow.com/questions/tagged/content-type)[x-www-form-urlencoded](https://stackoverflow.com/questions/tagged/x-www-form-urlencoded)

The first case is telling the web server that you are posting JSON data as in:

{"Name": "John Smith", "Age": 23}

The second case is telling the web server that you will be encoding the parameters in the URL:

Name=John+Smith&Age=23

What implications does it have on the server side. I see sites like stackoverflow & Twitter use x-www-form-urlencoded for AJAX requests like vote etc. The response sent back is JSON. I would think that it's better to have a symmetrical request/response pair i.e. both JSON.

– [user](https://stackoverflow.com/users/781695/user)

[Jul 20 '14 at 9:25](https://stackoverflow.com/questions/9870523/what-are-the-differences-between-application-json-and-application-x-www-form-url#comment38585196_9880122)

My understanding is using JSON as contentType helps when the data to be sent is more complex and involves a lot of hierarchy.. whereas form encoded is good to send simple params in url which can be read at the backend without to much code... I guess this answers the why part of it.

– [Ankit Srivastava](https://stackoverflow.com/users/919545/ankit-srivastava)

[Dec 13 '15 at 6:53](https://stackoverflow.com/questions/9870523/what-are-the-differences-between-application-json-and-application-x-www-form-url#comment56238992_9880122)

* @Medorator A late comment. Though for example, when you're sending a complex JSON object with an array of objects in it, using application/x-www-form-urlencoded would confuse the server (Elixir using Poison in my case) and result in some inappropriate parsing of the object (it somehow converted the nested array of objects to a map, instead of a list). Using application/json should be the correct choice in this case.

– [xji](https://stackoverflow.com/users/1460448/xji)

[Apr 2 '18 at 10:39](https://stackoverflow.com/questions/9870523/what-are-the-differences-between-application-json-and-application-x-www-form-url#comment86225774_9880122)

webRequest.ContentType = "application/x-www-form-urlencoded";

1. Where does **application/x-www-form-urlencoded**'s name come from?

If you send HTTP **GET** request, you can use query parameters as follows:

http://example.com/path/to/page**?name=ferret&color=purple**

The content of the fields is encoded as a query string. The application/x-www-form- urlencoded's name come from the previous url query parameter but the query parameters is in where the body of request instead of url.

The whole form data is sent as a long query string.The query string contains **name- value** pairs separated by **&** character

e.g. field1=value1&field2=value2

1. It can be **simple request** called simple - **don't trigger a preflight check**

Simple request must have some properties. You can look [here](https://dev.to/effingkay/cors-preflighted-requests--options-method-3024?signin=true) for more info. One of them is that there are only three values allowed for Content-Type header for simple requests

* + **application/x-www-form-urlencoded**
  + multipart/form-data
  + text/plain

3.For mostly flat param trees, application/x-www-form-urlencoded is tried and tested.

request.ContentType = "application/json; charset=utf-8";

1. The data will be [json](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON) format.

**axios** and **superagent**, two of the more popular npm HTTP libraries, work with JSON bodies by default.

{

"id": 1,

"name": "Foo",

"price": 123,

"tags": [

"Bar",

"Eek"

],

"stock": {

"warehouse": 300,

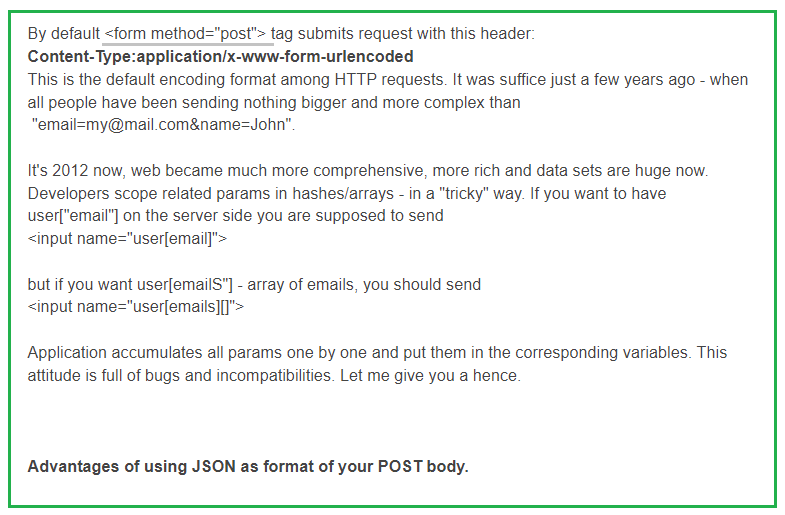
"retail": 20

}

}

1. **"application/json"** Content-Type is one of the **Preflighted requests**.

Now, if the request isn't **simple request**, the browser automatically sends a HTTP request before the original one by **OPTIONS** method to check whether it is safe to send the original request. If itis ok, Then send actual request. You can look [here](https://dev.to/effingkay/cors-preflighted-requests--options-method-3024?signin=true) for more info.

1. **application/json** is beginner-friendly. URL encoded arrays can be a nightmare!

One of the biggest differences between the two is that JSON-encoding the post usually preserves the data types of the values that are sent in (as long as they are valid JSON datatypes), whereas application/x-www-form-urlencoded will usually have all properties converted to strings.

For example, in the JSON-encoded post of:

{"Name": "John Smith", "Age": 23}

the server will most likely parse the Age property as the integer 23.

Whereas in

Name=John+Smith&Age=23

the server will most likely parse Age as the string "23".

Of course, if you are using other layers to parse these values and convert them to the appropriate types, this may not be an issue.