UPLOAD PROFILE PICTURE TUTORIAL

Composed by: Charlie Tan

Local Route: http://localhost:5000/api/upload_profile_picture (upload_profile_picture)

Deployed Route: https://kindling-lp.herokuapp.com/api/upload_profile_picture (upload_profile_picture)

EXPECTED INPUT FROM FRONTEND:

-This API works a bit differently from all the other API's thus far. Before, we've been expecting a JSON object, however for this API to work, we now expect input to be in the form of 'multipart form data'.

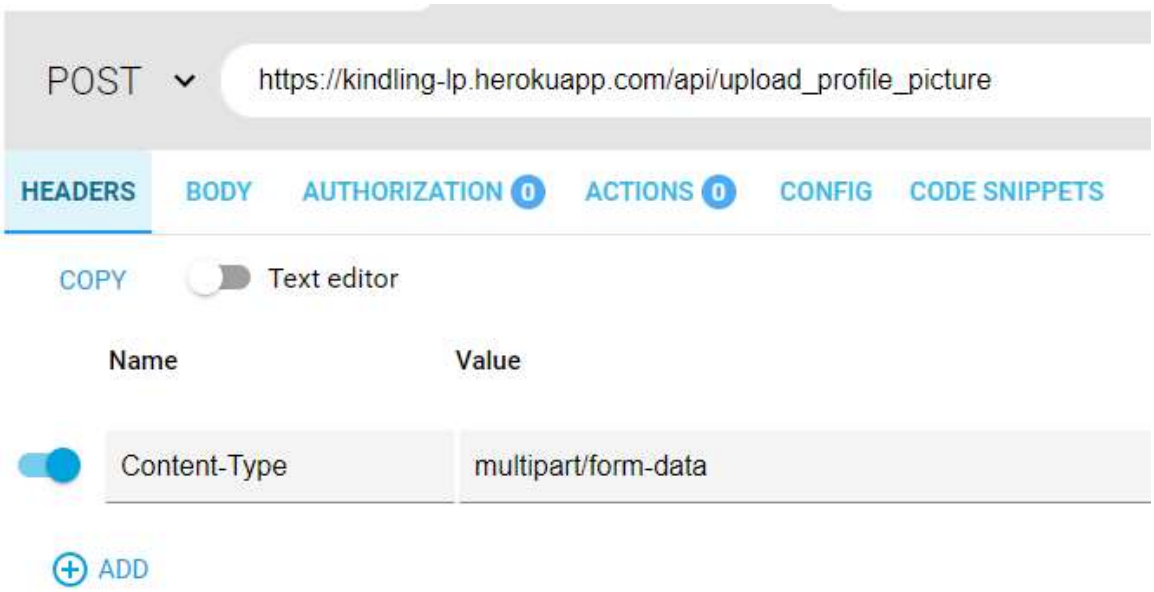
1) Image File - profile_picture
2) Text - email_str
3) Text - access_token_str

INPUT PROPERTIES EXPLAINED:

1) **profile_picture**: the image file that will be uploaded to our database.
2) **email_str**: the email string corresponding to the user for whom we are uploading 'profile_picture'.
3) **access_token_str**: this is the access token originally provided by the backend proving that you are who you say you are. For more information on access tokens, please reference the login API tutorial.

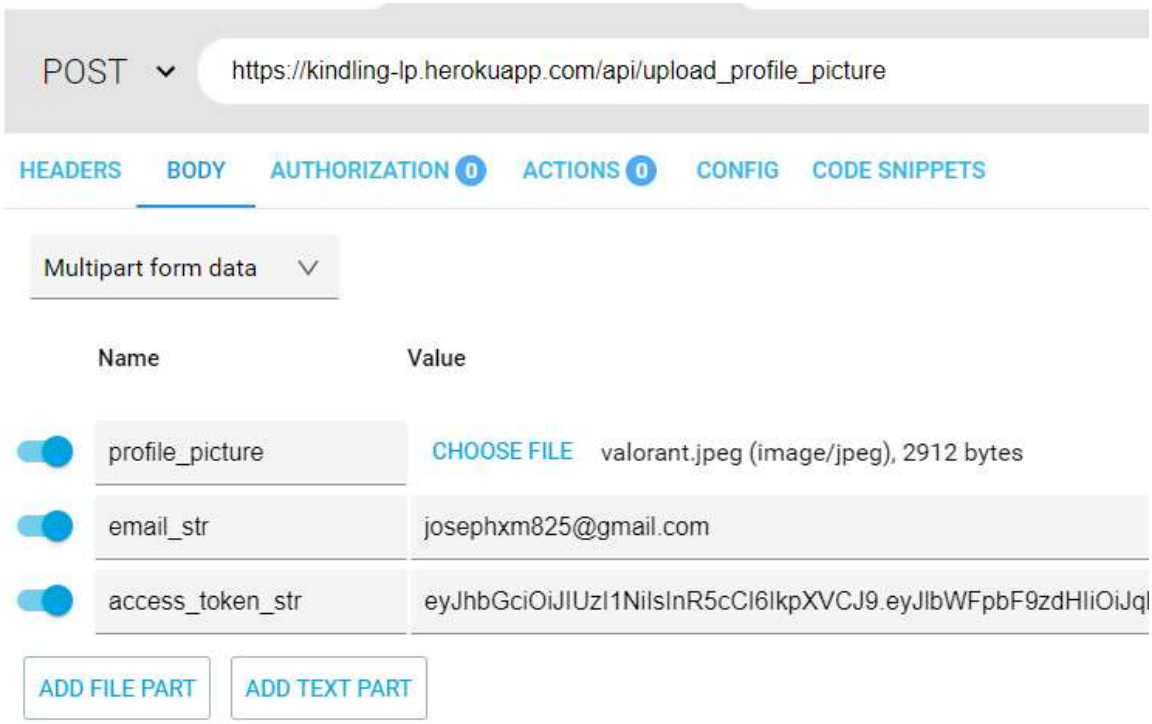
UNDERSTANDING MULTIPART FORM DATA:

-We are going to look at this from the point of view of using ARC (Advanced Rest Client) which is used to manually test our API's.



-In all prior API's, our header was set to 'Content-Type' - 'application/json' since we were obviously dealing with JSON objects as inputs. In the above picture, we can see now that we must set the header to 'Content-Type' - 'multipart/form-data'.

-What this allows us to do is send multiple files and text in a single POST request.



-We could send 'email_str' and 'access_token_str' in the form of a JSON object like we have been doing in the past, but what about 'profile_picture'? This is why we must use multipart form data.

IMPLEMENTING MULTIPART FORM DATA:
-OK, so I'll be the first to admit that I know very little about working with the frontend. Take what I say here with a grain of salt since I don't know how you guys have been sending information to the backend thus far.

-If you're thinking of using form tags to implement this, then be sure to set the encoding type property correctly like so:

```
<form action = "/(the desired route)" method = "POST" enctype = "multipart/form-data">
   <input (do what you need to do here) />
   ...
</form>
```

-If you don't set enctype to 'multipart/form-data', it is destined to fail!

-Again, I'm not an expert at the frontend. Do your own research. This is simply to give you a place to start.

EXPECTED OUTPUT FROM BACKEND:

```
{
 "success_bool" : some_boolean,
 "error_code_int" : some_integer,
 "refreshed_token_str" : some_string
}
```

-Unlike the input, the output is going to be the familiar JSON object we've been using this whole time.

OUTPUT PROPERTIES EXPLAINED:
1) **success_bool**: whether or not the 'profile_picture' was successfully uploaded to our database. 'true' if uploaded, 'false' otherwise.
2) **error_code_int:** in the case of not being able to upload successfully, an error code is provided to give additional information so that the user can possibly rectify their mistake.
3) **refreshed_token_str**: the refreshed access token that is provided by the backend upon a successful API call. For more information on access tokens, please reference the login API tutorial.

ERROR_CODE_INT EXPLAINED:

-There are several possible values for 'error_code_int':

-1 : bad token
0 : no error
1 : upload size too large
2 : user not found in database
3 : database write error
4 : database has too many pictures
5 : no image was uploaded

-Of the values shown above, I see two of them being useful for providing more information to the user: value 1, and value 4.

-For value 1 (upload size too large), a limit has been set for profile pictures as 3.2 MB.

-For value 4 (database has too many pictures), once our database is filled up to 80% capacity from profile pictures alone, no more will be accepted at that point (I guess we'd have to pay for more space if this ever happened, but for our project, we shouldn't worry).
