Design Document

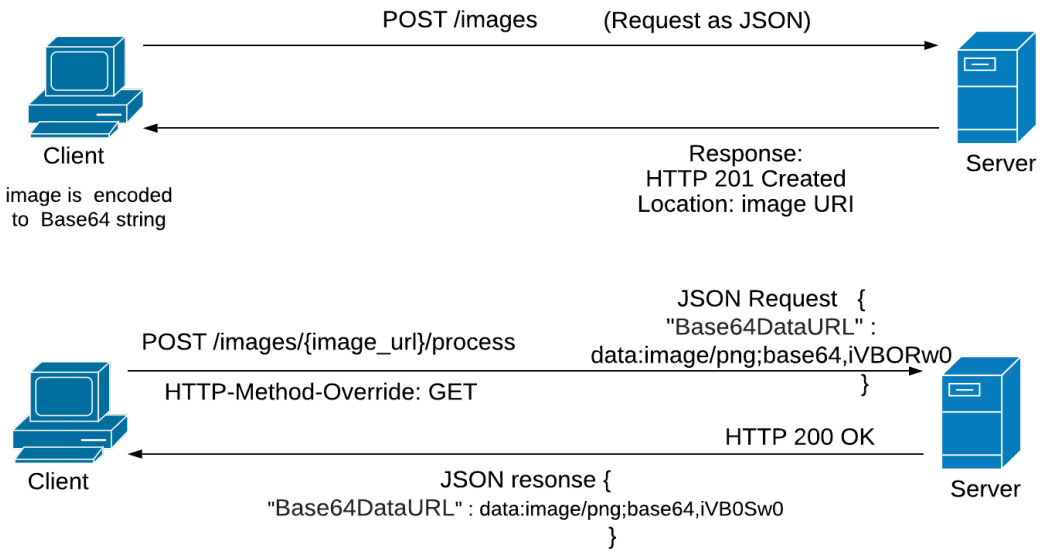
Architectural style:

The API will use REST architectural style and a JSON data format.

REST architecture is simple and easy to develop. REST web services allow for simple point to point communication over HTTP using JSON. REST style is scalable and will enable the system to handle many users at once.

The client will be able to submit a request to send an image from their local file system to the server, by making a POST call. The FileReader API will encode the image into a Base64 string representation which can then be inserted into JSON and communicated to the server. This will also allow client's browser to cache the image. The API will return dynamic image URLs.

In order to perform any of the required operations on the image at hand, a HTTP POST (with X-HTTP-Method-Override: GET) request will have to be performed on the endpoint. This is achieved by prefixing the image's URL with the API URL and passing the operation to be performed as query parameters. These operations can be applied programmatically in real time.



Methods and Endpoints:

POST /images - The client will be able to submit a request to upload an image from their filesystem. The image is encoded into Base64 and sent to the server in JSON format.

- Response status 201 created - The response will include a dynamic image URL in the Location header and metadata.
- Response status 404 - if the image was not found.

- POST /images/{image_url}/process – The dynamic image URL is manipulated by appending required operations as query parameters and sent over to the server. The web service will perform the given operations to create a resultant modified image. The Base64 encoded data URL is sent as a part of the request (JSON format) body. X-HTTP-Method-Override header is used with GET, as the URL length may end up being too long for a simple GET request. Moreover, GET is used as operations are performed on a single entity identified by the image_url and not on the whole resource, images. The modified image is sent over to the client as an encoded Base64 string within the response body. It can then be displayed in HTML.

    - Response status 200 OK - The response will be in JSON format and will include the modified image as an encoded Base64 string in the body.
    - Response status 400 - If the process/parameters or image URL are missing or incorrect.

Operations that can be performed with possible values for query parameters:

- Thumbnail:
    - POST /images/{image_url}?process={type}
      type: thumb (changes the dimensions to 160*160 pixels)

- Resize:
    - POST /images/{image_url}?process={type}&w={width}&h={height}
      type: res
      width: width in pixels
      height: height in pixels

- Rotate left:
    - POST /images/{image_url}?process={type}
      type: rotl, rotated clockwise by 90 degrees

- Rotate right:
    - POST /images/{image_url}?process={type}&d={degree}
      type: rotr, rotated counterclockwise by 90 degrees

- Grayscale:
    - POST /images/{image_url}?process={type}
      type: sat, reduces the saturation by a default degree of saturation
- Rotate:
    - POST /images/{image_url}?process={type}&d={degree}
      type: rot
      d: 0-359 degrees, rotated counterclockwise
- Flip:
    - POST /images/{image_url}?process={type}&a={axis}
      type: flip
      a: h (horizontal) and or v (vertical)

Web Service Design Patterns:

- Web Service API Styles:
  - Resource API: Since resource APIs use HTTP as a complete application protocol, clients will be able to simply use these standards defined in the HTTP specification without having to create responses for each service outcome. For instance, issue a POST to retrieve a modified image, POST to create a resource (image) and receive a standard status code as response.

- Client-service Interaction Styles:
  - Request / response: It is the simplest way for the web service to receive a request and an associated resource (image) and perform functions like rotate and return the desired outcome.

  - Linked Service: Since the project problem requires multiple manipulation operations to be performed on the same image such that we end up with one modified image, applying this design pattern will allow the service to include the addresses of other operations in each response. Clients will be able to parse responses to discover subsequent service URIs.

- Request and Response Management:
  - Service Controller: When the server receives a request, the framework will invoke the appropriate handler by evaluating the request against URI templates. This pattern works in this case by selecting the service based on the client's preferred message header values, or message body content.

- Web Service Implementation:
  - Transaction Script: The web services can be implemented as Transaction Scripts that will include all of the code for image manipulation allowing developers to easily implement the web service logic.

- Web Service Infrastructures:
  - Service Connector: Since the web service requires images to be encoded into Base64 for communication via JSON, this design pattern allows for connectors to encapsulate the generic communications-related logic.

  - Service Descriptor: Since this approach creates a standardized description of related services that identifies URIs and logical operations. This can be applied to our web service enabling prefixing the image's URL with the API URL and passing the operations to be performed (like crop) as URL parameters.

Trade off:

- Base64 encoding makes file sizes roughly 33% larger than their original binary representations, which means more data down the wire
- Data URIs aren't supported on IE6 or IE7.
- Base64 encoded data may possibly take longer to process than binary data.

References:

- https://docs.imgix.com/apis/url
- https://blog.online-convert.com/basic-image-editing-via-api/
- https://stackoverflow.com/questions/34485420/how-do-you-put-an-image-file-in-a-json-object/34485762