SDK specification

General Guidelines

camelCasing or snake_casing to be followed for all parameters depending upon the implementation language
Angular, React or any other SDK that should use Component type of implementation then the parameters should be the same but should be used as a Component.

Client-Side SDK

1. **Initialization**
    a. Initialization should throw error if parameter named “privateKey” is passed as a parameter
    { message : “privateKey should not be passed on the client side”, help : “” }
    b. Initialization error when publicKey or urlEndpoint is missing
    { message : “Missing publicKey/urlEndpoint during initialization”, help : “” }
    c. Initialization error when transformationPosition is not valid (path or query)
    { message : “Invalid transformationPosition parameter”, help : “” }

```
imagekit = new ImageKit({
  "publicKey" : "your_public_api_key", //required
  "urlEndpoint" : "https://ik.imagekit.io/your_imagekit_id", //required
  "transformationPosition" : 'path'  //optional - default will be 'path', accepts 'path' or 'query'. If src parameter is being used, then always force the addition of transformation parameters in query
  "authenticationEndpoint" : '' //Server API Endpoint that will provide auth token and signature, optional, required for only upload
});
```

2. **URL Generation**

Can be done in two ways - using the 'path' of the image or using the complete image URL that is stored in your DB as 'src'

a. This method should check that if both path and src are present, then path is preferred.
b. If both of them are missing, return empty URL
c. This method should handle that both path and src can contain query parameters themselves

```
imagekit.url({
  urlEndpoint : "https://ik.imagekit.io/demo/pattern", //optional, will use the urlEndpoint specified at initialization if not specified
  path : "path/to/my/image.jpg", //path of the image
  transformation : [{ //array of objects, transformations to be applied - optional
    "height" : "300",
    "width" : "200"
  }, {
    "rotate" : "90"
  }],
  transformationPostion : "query", //or 'path' - optional. If src parameter is being used, then always force the addition of transformation paramters in query
  queryParameters : { //any other query parameters that need to be added to the URL - optional
    "v" : "123123"
  }
}); // = https://ik.imagekit.io/demo/pattern/path/to/my/image.jpg?tr=w-300,h-200:rt-90&v=123123


//if the customer has the entire image url stored in their database and just wants to transform the image using the SDK
imagekit.url({
```

```
  src : "https://ik.imagekit.io/demo/medium_cafe_B1iTdD0C.jpg", //full image URL
  transformation : [{ //array of transformations to be applied - optional
    "height" : "300",
    "width" : "200"
  }, {
    "rotate" : "90"
  }],
  transformationPostion : "query", //or 'path' - optional. If src parameter is being used, then always force the addition of transformation paramters in query
  queryParameters : { //other query parameters needed in the URL - optional
    "v" : "123123"
  }
}) //= https://ik.imagekit.io/demo/medium_cafe_B1iTdD0C.jpg?tr=w-300,h-200:rt-90&v=123123;
```

3. **Client-side Upload**
   a. The upload function will check if 'authenticationEndpoint' was provided in initialization. If not provided, it results in an error
   b. Otherwise it gets the authorization credentials from this endpoint, uses it in the request if successfully fetched. The error and result are populated from the authentication endpoint
   c. The 'expire' timestamp will be automatically created by SDK for 1 minute default time
   d. Handle no 'file', no 'fileName' case
   e. Optional parameters that can be passed along with file and fileName
   useUniqueFileName
   tags
   folder
   isPrivateFile

customCoordinates
responseFields

```
imagekit.upload({
  file : <url|base_64|binary>, //required
  fileName : 'your_file_name'  //required
}, function(error, result) {


})
```

Server-Side SDK

1. **Initialization**
   a. Throw InitializationError when publicKey or urlEndpoint or privateKey is missing
   { message : “Missing publicKey or privateKey or urlEndpoint during ImageKit initialization”, help : “” }
   b. Initialization error when transformationPosition is not valid (path or query)
   { message : “”, help : “” }

```
//Server-side Initialization
imagekit = new ImageKit({
  "publicKey" : "my_api_key", //required
  "privateKey" : "my_private_key", //required
  "urlEndpoint" : "https://ik.imagekit.io/imagekitId", //requi
red
  "transformationPosition" : 'path'  //optional - default will
be 'path'. If src parameter is being used, then always force t
he addition of transformation paramters in query
});
```

2. **URL Generation**

Can be done in two ways - using the 'path' of the image or using the complete image URL that is stored in your DB as 'src'

a. This method should check that if both path and src are present, then path is preferred.
b. If both of them are missing, return empty URL
c. This method should handle that both path and src can contain query parameters themselves

```
//Image URL Generation
//Can be done in two ways - using the image path or using the complete image URL that is stored in your DB
imagekit.url({
  //overrides the defaul URL Endpoint for this particular image
  urlEndpoint : "https://ik.imagekit.io/demo/pattern",
  path : "path/to/my/image.jpg",
  transformation : [{ //array of transformations to be applied
    "height" : "300",
    "width" : "200"
  }, {
    "rotate" : "90"
  }],
  transformationPostion : "query", //or 'path'. If src parameter is being used, then always force the addition of transformation paramters in query
  queryParameters : { //any other query parameters that need to be added to the URL
    "v" : "123123"
  }
}); // = https://ik.imagekit.io/demo/pattern/path/to/my/image.jpg?tr=w-300,h-200:rt-90&v=123123
```

```
//if the customer has the entire image url stored in their dat
abase and just wants to transform the image using the SDK
imagekit.url({
  src : "https://ik.imagekit.io/demo/medium_cafe_B1iTdD0C.jp
g",
  transformation : [{ //array of transformations to be applied
    "height" : "300",
    "width" : "200"
  }, {
    "rotate" : "90"
  }],
  transformationPostion : "query", //or 'path'. If 'src' is sp
ecified, then always force addition of transformation paramete
rs as query
  queryParameters : {
    "v" : "123123"
  }
}) //= https://ik.imagekit.io/demo/medium_cafe_B1iTdD0C.jpg?tr
=w-300,h-200:rt-90&v=123123;
```

3. **Signed URL generation**
   a. The same `url` method accepts a new parameter
   b. `signed` that should be `true` (boolean) for generating a signed URL. If signed parameter is not `true` or `false` boolean, then it should be considered as `false`
   c. `expireSeconds` is optional. This gets converted to a timestamp in the backend for these many seconds before the current timestamp. If this parameter is specified then the output URL contains `ik-t` parameter (unix timestamp seconds when the URL expires) and the signature contains the timestamp for computation. If not present, then no ik-t parameter and the value `9999999999` is used.

```
imagekit.url({
  //overrides the defaul URL Endpoint for this particular imag
e
  urlEndpoint : "https://ik.imagekit.io/demo/pattern",
  path : "path/to/my/image.jpg",
  transformation : [{ //array of transformations to be applied
    "height" : "300",
    "width" : "200"
  }, {
    "rotate" : "90"
  }],
  transformationPostion : "query", //or 'path'
  queryParameters : { //any other query parameters that need t
o be added to the URL
    "v" : "123123"
  },
  "signed" : true,
  "expireSeconds" : 300 //optional - the relative timestamp af
ter which the SDK shouldexpire the URLs (so that you do not ha
ve to generate the timestamp and then add 300 seconds)
})
```

4. **Signature Generation**
   a. returns { token : <token>, timestamp : <timestamp_in_seconds>, signature : <signature> } in the output
   b. `token` input parameter is optional

```
var authenticationParameters = imagekit.getAuthenticationParam
eters(token, timestamp);


//pseudo code for getAuthenticationParameters()
```

```
function getAuthenticationParameters(token, timestamp) {
  if(!token) token = uuid.v4();
  if(!timestamp) timestamp = parseInt(new Date().getTime() / 1000, 10) + 2400;
  signature = hmac_sha1_of(token+timestamp);
  return {token : token, timestamp : timestamp, signature : signature};
}
```

5. **Upload**
   a. The upload function will automatically generate the authorization header and issue the upload request
   b. Handle no 'file', no 'fileName' case and no data at all case
   { message : “Missing data for upload”, help : “” }
   { message : “Missing file parameter”, help : “” }
   { message : “Missing fileName parameter”, help : “” }
   c. Optional parameters that can be passed along with file and fileName
   useUniqueFileName
   tags
   folder
   isPrivateFile
   customCoordinates
   responseFields

```
imagekit.upload({
  file : <url|base_64|binary>, //required
  fileName : "my_file_name",   //required
}, function(error, result) {});
```

6. **List Files**
   a. Accepts all optional parameters as given in the doc
   b. Automatically generates auth header
   c. If you do not want to pass any parameters, pass empty {}
   d. Handle invalid listingOptions case. It can only be an object

{ message : "Missing options for list files", help : "If you do not want to pass any parameter for listing, pass an empty object" }

```
imagekit.listFiles({}, function(error, result) { })
```

7. **Get File Details**
   a. Handle no file ID case
      { message : “Missing file ID parameter”, help : “” }

```
imagekit.getFileDetails("the_file_id", function(error, result) { });
```

8. **Get Metadata**
   a. Handle no file ID case
      { message : “Missing file ID parameter”, help : “” }

```
// Get metadata using fileId of a file already uploaded in the media library
imagekit.getFileMetadata("the_file_id", function(error, result) { });

// Get metadata of an image using a remote URL accessible through the same ImageKit account
imagekit.getFileMetadata("https://ik.imagekit.io/your_imagekit_id/image.jpg", function(error, result) { });
```

9. **Update File Details**
   a. Handle no file ID case
      { message : “Missing file ID parameter”, help : “” }
   a. Handle no update parameters case
      { message : “Missing file update parameters”, help : “” }
   a. Handle tags - should be array or null (to unset tags)
      { message : "Invalid tags parameter for this request", help : "tags should be passed as null or an array like ['tag1', 'tag2']" }
   a. customCoordinates - can be null, or string of `x,y,width,height`. Check validity of x,y,width,height format using regex.

{ message : "Invalid customCoordinates parameter for this request", help : "customCoordinates should be passed as null or a string like 'x,y,width,height'" }

```
imagekit.updateFileDetails("the_file_id", {
  tags : ["tag1", "tag2"],
  customCoordinates : "10,10,100,100"
}, function(error, result) {});
```

10. **Delete File**
    a. Handle no file ID case
    { message : “Missing file ID parameter”, help : “” }

```
imagekit.deleteFile("the_file_id", function(error, result) {
})
```

11. **Purge Cache**
    a. Handle no URL case
    { message : “Missing URL parameter for cache purge”, help : “” }

```
imagekit.purgeCache("full_url", function(error, result) { });
```

12. **Get Purge Cache Status**
    a. Handle no request ID case
    { message : “Missing request ID parameter”, help : “” }

```
imagekit.getPurgeCacheStatus("requestId", function(error, result) { });
```

If the SDK validation fails, then return the
{ message : “Invalid transformationPosition parameter”, help : “” }
type of object in in error, result should be `null`

All the above API implementations will return the result as it is from the server without modification.

If the API fails, error should be set as the response received from the server, result should be `null`.

If the API succeeds, error should be set as `null`, success should be set as the response received from the API

Test Cases

1. For image URLs, success cases with all different parameters
2. For all APIs, test cases for a success response

Demo

Include a simple demo that implements the upload API and the URL generation