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;
```

Note: The client-side SDK should add a query parameter in the image URL. Parameter name should be `ik-sdk-version` with the value `SDK_NAME-VERSION` e.g. `android-1.0.0`, `ios-1.0.1` or `react-1.0.2` etc.

Note: The SDK should handle the presence and absence of trailing slash in `urlEndpoint` parameter. Also the SDK should handle the presence and absence of leading slash in `path` parameter. The SDK should not add double slash e.g. `{urlEndpoint}//{path}`

3. **Client-side Upload (please use this endpoing - https://upload.imagekit.io/api/v1/files/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

Upload API Endpoint to be used in client-side SDKs - https://upload.imagekit.io/api/v1/files/upload

```
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 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'. 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 database and just wants to transform the image using the SDK
imagekit.url({
  src : "https://ik.imagekit.io/demo/medium_cafe_B1iTdD0C.jpg",
  transformation : [{ //array of transformations to be applied
    "height" : "300",
    "width" : "200"
  }, {
    "rotate" : "90"
  }],
  transformationPostion : "query", //or 'path'. If 'src' is specified, then always force addition of transformation parameters 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.getAuthenticationParameters(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

Upload API endpoint to be used in server side SDK -
https://upload.imagekit.io/api/v1/files/upload

```
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 (with file ID or using URL)**
   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 thro
ugh 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. **Delete file (bulk)**
    a. Handle no file ID case
       { message : “Missing file ID parameter”, help : “” }

```
imagekit.bulkDelete(["fileId1","fileId2"], function(error, res
ult) { })
```

1. **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, resu
lt) { });
```

Utility Functions

Authentication parameter generation

In case you are looking to implement client-side file upload, you are going to need a token, expiry timestamp and a valid signature for that upload. The SDK provides a simple method that you can use in your code to generate these authentication parameters for you.

Note: The Private API Key should never be exposed in any client-side code. You must always generate these authentication parameters on the server-side

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

Returns

```
{
    token : "unique_token",
    expire : "valid_expiry_timestamp",
    signature : "generated_signature"
}
```

Both the `token` and `expire` parameters are optional. If not specified the SDK uses the **uuid** package to generate a random token and also generates a valid expiry timestamp internally. The value of the `token` and `expire` used to generate the signature are always returned in the response, no matter if they are provided as an input to this method or not.

Distance calculation between two pHash values

Perceptual hashing allows you to constructing a hash value that uniquely identifies an input image based on the contents of an image. **ImageKit.io metadata API** returns the pHash value of an image in the response. You can use this value to find a duplicate (or similar) image by calculating distance between pHash value of two images.

This SDK exposes `pHashDistance` function to calcualte distance between two pHash values. It accepts two pHash hexadecimal strings and returns a numeric value indicative of the level of difference between the two images.

```
const calculateDistance = () => {
    // asynchronously fetch metadata of two uploaded image fil
es
    // ...
    // Extract pHash strings from both: say 'firstHash' and 's
econdHash'
    // ...
    // Calculate the distance between them:
    const distance = imagekit.pHashDistance(firstHash, secondH
ash);
    return distance;
}
```

Distance calculation examples

```
imagekit.pHashDistance('f06830ca9f1e3e90', 'f06830ca9f1e3e9
0');
```

```
// output: 0 (same image)

imagekit.pHashDistance('2d5ad3936d2e015b', '2d6ed293db36a4f
b');
// output: 17 (similar images)

imagekit.pHashDistance('a4a65595ac94518b', '7838873e791f840
0');
// output: 37 (dissimilar images)
```

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

URL-generation test cases (client-side SDKs and server-side SDK)

1. Overriding `urlEndpoint` parameter. Pass a `urlEndpoint` value which is different from what you have used during SDK initialization and see if the url returned is using this new parameter.
2. Presence and absence of trailing slash in `urlEndpoint` should not result in double slash (//) in the returned url. Please make sure to use RegEx or begins with condition to test this. Otherwise test will always pass even if the url has double slash.

3. Presence and absence of leading slash in `path` parameter should not result in double slash (//) in the returned url. Please make sure to use RegEx or begins with condition to test this. Otherwise test will always pass even if the url has double slash.
4. If a new transformation parameter (for which SDK don't have a mapping in declared constants) is passed then it should come in URL as it is.
5. The url returned should have `ik-sdk-version` parameter with the value `SDK_NAME-VERSION` e.g. `android-1.0.0` , `ios-1.0.1` or `react-1.0.2` etc.
6. `tranformationPosition` should be set to `query` and check if now transformations are in query parameter or not.
7. Check chained transformations.
8. Check if parameters passed to `queryParameters` are being added to the returned url.
9. Pass `src` instead of `path` and check url.
10. Pass `src` with a query parameters instead of `path` and pass some other values in `queryParameters` to check if query parameters are merged correctly. Look out for double && or double ?? in url returned.
11. Pass `src` instead of `path` and some transformation parameters and set `transformationPostion` to `path`. SDK should always add transformation as query parameter in this case no matter what is the value of `transformationPostion` parameter.

URL-generation test cases (server-side SDK only)

1. Generate a signed URL with default expireSeconds value.

```
var imagekit = new ImageKit({
    publicKey : "public_key_test",
    privateKey : "private_key_test",
    urlEndpoint : "https://test-domain.com/test-endpoint"
})
var imageURL = imagekit.url({
    path : "/test-signed-url.png",
    signed : true,
    transformation: [{
        width : 100
```

```
    }]
});

// Pur a assert condition on below URL (exact match). Hardcode
it in test cases
console.log(imageURL)
// https://test-domain.com/test-endpoint/tr:w-100/test-signed-
url.png?ik-s=eaf466ccbf4d2573c71bca2ab7c4c2f2f47ddc1f
```

2. Generate a singed URL and pass expiredSeconds value

```
var imagekit = new ImageKit({
    publicKey : "public_key_test",
    privateKey : "private_key_test",
    urlEndpoint : "https://test-domain.com/test-endpoint"
})
var imageURL = imagekit.url({
    path : "/test-signed-url.png",
    signed : true,
    expireSeconds : 100,
    transformation: [{
        width : 100
    }]
});

// Assert that value of imageURL should have "ik-t"
// No need to match the exact URL because timestamp will keep
changing and we would not know the final signature. We just ne
ed to make sure ik-t is coming in URL in this case.
```

Upload API (client and server side SDKs)

1. Check if absence of tags is not resulting in sending undefined or null values. Please note that everything is sent as strings to the API so the undefined doesn't

mean undefined for backend.
2. Check if absence of isPrivateFile is not resulting in sending undefined or null values. Please note that everything is sent as strings to the API so the undefined doesn't mean undefined for backend.
3. Check if absence of customCoordinates is not resulting in sending undefined or null values. Please note that everything is sent as strings to the API so the undefined doesn't mean undefined for backend.
4. Check if absence of useUniqueFileName is not resulting in sending undefined or null values. Please note that everything is sent as strings to the API so the undefined doesn't mean undefined for backend.
5. Check if tags are being passed properly.
6. Check if absence of folder is not resulting in sending undefined or null values. Please note that everything is sent as strings to the API so the undefined doesn't mean undefined for backend.
7. Check if tags are being passed properly.
8. Check if customCoordinates are being passed.
9. Check if fileName can be set.
10. Check if responseFields can be set.
11. Check if useUniqueFileName can be set.
12. Check if file parameter can accept binary|base64|url type data structures.

Other API and Utility function test cases (only server side)

1. Check the listFiles function is accepting and passing all the options as query parameter in the API.
2. Basic unit test case for getFileDetails
3. Basic unit test case of getFileMetadata. It should accept url parameter
4. Basic unit test case of getFileMetadata. It should accept fileId parameter
5. Basic unit test case for purgeFile. It should accept a single string parameter which is URL of the resource that needs to be purged.
6. fileUpdate test - Send the update object JSON as it is to the backend.
7. DeleteFile test - Should accept the fileId parameter
8. BulkFileDelete test - Should accept the array of fileId.
9. Purge cache status check - Should accept only one parameter i.e. purge requestId.
10. Check the auth parameters (pass token and timestamp)

```
var imagekit = new ImageKit({
```

```
    publicKey : "public_key_test",
    privateKey : "private_key_test",
    urlEndpoint : "https://test-domain.com/test-endpoint"
})

var authenticationParameters = imagekit.getAuthenticationParam
eters("your_token",1582269249);

/*
  Assert that authenticationParameters is same as below JSON.
Hard code it in test cases
  {
    token: 'your_token',
    expire: 1582269249,
    signature: 'e71bcd6031016b060d349d212e23e85c791decdd'
  }
*/
```

3. The pHashDistance function test between “33699c96619cc69e” and “968e978414fe04ea”

```
var imagekit = new ImageKit({
    publicKey : "public_key_test",
    privateKey : "private_key_test",
    urlEndpoint : "https://test-domain.com/test-endpoint"
})

var pHashDistance = imagekit.pHashDistance("33699c96619cc69
e","968e978414fe04ea");
console.log(pHashDistance);
// 30
// Assert that value of pHashDistance should be 30.
```

```
var pHashDistance = imagekit.pHashDistance("33699c96619cc69e","33699c96619cc69e");
console.log(pHashDistance);
// 0
// Assert that value of pHashDistance should be 0.
```

Demo

Include a simple demo that implements the upload API and the URL generation (signed url, chained transformations)