Integration Manual

Version: 2019-10.a

Instant Design Tool

https://www.instantdesigntool.com

INDEX

1 GETTING STARTED

Note: your Design Tool comes standard with a free backend provided by us. A shopping cart and order backoffice are included. Only follow this manual if you want to do a custom integration.

To disconnect your Design Tool from our backend and unlock the developer settings, first log into the IDT management panel and follow the prompts on the "Developer" settings page.

1.1 INTRODUCTION

This document explains how to set up a custom integration between an Instant Design Tool and your system. It covers user identification, ordering, and PDF/preview generation.

In short: you'll be asked to add a few API endpoints to your system, which conform to the specifications in this document. The Design Tool will POST form data to these endpoints during specific events. Your system will respond in a predefined format and may take further action later if needed.

- You can choose your own custom URL for each endpoint
- Chapter 5 provides a checklist of all the technical info we'll need

If you use a closed third-party system and are unable to add some of the API endpoints we need, our system is designed to support tailored solutions to an extent. Please contact us for more info.

1.2 CREATING A DNS RECORD

The first thing you'll need is a subdomain for your Design Tool instance. For example: "designer.your-site-name.com". Note that this *must* be a subdomain of your main website for cookie-related features to work correctly (see ch. 2). Point the subdomain at our webserver by making a CNAME-type DNS record with the following value: "app.instantdesigntool.io"

Note: you'll be prompted to do this when unlocking the developer settings.

1.3 DEFINING A SERVER-TO-SERVER API KEY

Every request sent by the Design Tool will include an "apiKey" field, which you can use to verify the legitimacy of the request. You can provide your own API key, or we can generate one for you.

2 USER IDENTIFICATION

2.1 OVERVIEW

Your main website probably already has, or is otherwise likely planned to have, a cookie-based login system. The Design Tool is designed to seamlessly integrate with any such login system, so your users can securely save their projects to their account.

Note: when we say "seamlessly integrate", we mean exactly that. Customers will not have to log in multiple times during their journey, and there is no separate *"Instant Design Tool account"*. Your *existing* account system can be connected directly with our Design Tool.

Because the Design Tool runs on a subdomain of your main website, it can be given access to your main website's login cookie. Obviously, the Design Tool won't be able to parse the (likely encrypted) contents of your cookie, but it doesn't need to: instead, it will ask your system to handle that.

Specifically, you will need to add an endpoint in your system that takes a login cookie as input, and returns a unique user ID as output (assuming the cookie is valid). The Design Tool will be calling this endpoint frequently to identify the current user whenever needed (see 0).

If a user performs an action that requires login, and the Design Tool detects that the user is not logged in to your website, the Design Tool will display its built-in login form. Submissions of this form will also be handled by an endpoint in your system (see 2.3). Actions that require login include:

- Manually saving a project / opening a previously saved project
- Ordering a project (by default, but the login requirement **can** be turned off here)

Prerequisites for login integration to work are described in 2.2.

For most systems, just passing around a single login cookie is sufficient. But if your system requires more, we can enable "user impersonation mode" for server-to-server calls. When enabled, all requests from our server will appear and react as if they came directly from the user:

- Every request sent to your endpoints will include all cookies available on the Design Tool domain
- All cookies in your endpoint responses will be passed on to the user's browser
- The User-Agent header will match the user's browser

Please contact us for details if you need to enable this mode.

2.2 PREREQUISITES

For login integration to work, your login cookie's "domain" attribute must match both your Design Tool instance and your main website. For example, if your main website is "www.example.com" and your Design Tool is located at "editor.example.com", then the cookie domain must be ".example.com".

By default, we expect the login cookie name to be "auth", but this can be customized. If you use a different login cookie name, change the cookie name setting via your management panel.

2.3 ENDPOINT: POST – LOG IN USER

As described in 2.1, the Design Tool may sometimes prompt the user to log in. Once a user submits the login form, the Design Tool calls on your server to validate the credentials and, if they are valid, set a login cookie. (So, probably what your existing login form already does.)

Below are the fields we'll POST to your endpoint, in the "application/x-www-form-urlencoded" data format. So in PHP, for example, you can access these fields via the $_POST variable. Similarly, in ASP.NET, you can use the built-in model binding.

Request		
Property	**Type**	**Description**
`apiKey`	String (*)	See 1.3
`email`	String (1-128)	The e-mail address that was entered
`password`	String (1-128)	The password that was entered

The HTTP response of your system must have a status code in the 2xx range, and a body in the JSON data format. It only has to include the status of the login attempt and, on success, the resulting login cookie. We'll pass this cookie on to the browser, so that the customer is now logged in.

Response		
Property	**Type**	**Description**
`status`	String (*)	One of "InvalidCredentials" or "Success"
+ On success, your regular login cookie (as described in 2.2)		

EXAMPLE RESPONSE TO INVALID CREDENTIALS

Relevant headers
`Content-Type: application/json`
JSON body
`{"status": "InvalidCredentials"}`

EXAMPLE RESPONSE TO SUCCESFUL LOGIN

Relevant headers
`Content-Type: application/json` `Set-Cookie: auth=…; domain=.example.com; path=/; Secure; HttpOnly`
JSON body
`{"status": "Success"}`

2.4 ENDPOINT: POST – RETRIEVE USER ID

As described in 2.1, the Design Tool depends on your system to identify the current user. Whenever the user performs an action that requires a known identity, the Design Tool sends a request to your server to determine the user ID associated with their login cookie.

The call to this endpoint only contains the usual "apiKey" field (as always, in the "application/x-www-form-urlencoded" format). It also includes the login cookie (2.2) in the "Cookie" HTTP header, which is how your system will be able to detect the current user ID.

Request		
Property	**Type**	**Description**
`apiKey`	String (*)	See 1.3
+ Current login cookie		

The HTTP response of your system must have a status code in the 2xx range, and a body in the JSON data format. The body must include a 'userId' field: this is the unique identifier of the currently logged in user (between 1 and 36 characters). All projects of the user will be tied to this identifier.

If the supplied cookie is invalid (e.g. expired or forged), your system should return a null ID, or simply a blank response, and the Design Tool will reset the user's login status. Examples:

EXAMPLE RESPONSES TO VALID COOKIE

JSON body
`{"userId": "1459035312"}`

JSON body
`{"userId": "149e293e-442e-4d9b-81b9-4da1bf930a21"}`

EXAMPLE RESPONSES TO INVALID COOKIE

JSON body
`{"userId": null}`

Empty body

3 ORDERING

3.1 OVERVIEW

When a user clicks "Order", we first create an immutable snapshot of the project. We then push info about that snapshot to your server in the background (details in 3.2), such as:

- A product ID, user ID and project ID
- A file generation URL (for requesting a PDF and/or image-based preview)

This info is typically stored as a shopping cart item. After the above info is pushed to your server, the user will be redirected to your checkout process. From there, your system takes over, and you can later call on the Design Tool to provide preview images and PDF files (see ch. 4).

3.2 ENDPOINT: POST – PUSH SNAPSHOT

As described in 3.1, when a user clicks "Order", we first push project data to your server. Below are the fields we'll POST (again formatted "application/x-www-form-urlencoded"). You may not need to store and/or use all of these fields, depending on your use case.

Request		
Property	**Type**	**Description**
`apiKey`	String (*)	See 1.3
`title`	String (1-128)	The title of the project, either a default or user input.
`snapshotId`	String (32)	Identifies the snapshot being pushed.
`projectId`	String (32)	Identifies the source project. Note that multiple snapshots may be pushed with the same project ID.
`userId`	String (1-100)	Identifies the user. Retrieved from your system (see 0). If you disable the login requirement, this field may be empty.
`productId`	String (1-100)	Identifies the product type. Configurable per product, can be set up to match the SKUs in your system.
`pageCount`	Integer	The number of pages in the "content" PDF file.
`outputRequestUrl`	String (*)	HTTP POST URL for requesting file output (see 4)
`thumbnailUrl`	String (*)	HTTP GET URL for a 250px thumbnail image
`editUrl`	String (*)	An HTTP link the user can open to edit the project. Note that changes are not reflected in the file output of this snapshot.

Note again that if "user impersonation mode" is active, **every** call made to your server will also include all cookies available to the Design Tool subdomain.

The HTTP response of your system must have a status code in the 2xx range, and a body in the JSON data format. After receiving your HTTP response, we'll redirect the user to a URL of your choosing. The optional 'token' field in your response will simply be directly appended to that URL. If needed, you can use this feature to dynamically redirect the user based on product type, for example.

Response		
Property	**Type**	**Description**
`token`	String (0-128)	An optional token to append to your redirection URL.

3.3 OPTIONAL ENDPOINT: POST – MARK PROJECT AS DELETED

When a user clicks the 'Delete' button for their project, you may want to invalidate any links back to said project in the Design Tool that you've saved during 3.2. If you don't keep such links, you don't need to implement this endpoint.

The body of this request only contains the '`ApiKey`' field. The project ID is appended directly to the URL. So you'll need and endpoint URL like the following:

- https://www.example.com/api/mark-as-deleted/{projectId}
- https://www.example.com/api/mark-as-deleted?projectId={projectId}

The HTTP response of your system must have a status code in the 2xx range, and the body can be empty.

4 OUTPUT FILES

4.1 OVERVIEW

Our servers can generate certain "output files" from the snapshot you received in 3.2. We call this process "rendering". There are two types of output files that can be rendered:

- PDFs (high DPI, you'll need this for printing)
- JPEG- or PNG-based previews (lower DPI, suitable for online viewing)

Rendering is an expensive process, especially at high DPI. So even though our rendering servers are massively parallelized, there is a small delay before the files become available. Note:

- PDF files are only generated on-demand (i.e. at your request, see 4.2)
- Preview files are always generated (but making use of them is optional)

Once output files for a snapshot are available, the Design Tool can call a webhook on your server, so you don't have to poll us periodically. See 4.3 for details.

If you intend to use a partnered print-on-demand service API for order fulfilment, you may skip the PDF-related parts of this chapter (such as 4.2). Instead, when you submit an order to their API, include the `outputRequestUrl` (from 3.2) like so:

```
"files": {
  "output": "outputRequestUrl value goes here"
}
```

The print-on-demand REST API will handle the rest automatically. You can simply omit the regular `"content"` / `"cover"` properties.

4.2 Requesting PDF output (POST to our API)

After the user has placed their order, you'll probably need the PDF file. Previews are free to generate, but PDFs are not. So you must explicitly request PDF output files once they are needed.

> As detailed in 4.1, if you use our print-on-demand service, you can skip to 4.3. We handle this chapter for you. But if you handle fulfilment yourself, or via some other party, read on.

To request PDF output for a snapshot, send a simple HTTP POST to the `outputRequestUrl` that you received in 3.2. The request you should send is very simple:

- The body should be just: `outputType=pdf`
- The `Content-Type` header should be `application/x-www-form-urlencoded`

.NET code sample using `HttpClient`:

```
var response = await httpClient.PostAsync(outputRequestUrl,
    new StringContent("outputType=pdf", Encoding.UTF8,
        "application/x-www-form-urlencoded"));

if (!response.IsSuccessStatusCode)
    throw new Exception("Output request failed: "
        + await response.Content.ReadAsStringAsync());
```

PHP code sample using `cURL`:

```
$ch = curl_init($output_request_url);
curl_setopt($ch, CURLOPT_POST, 1);
curl_setopt($ch, CURLOPT_POSTFIELDS, 'outputType=pdf');
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
$response = curl_exec($ch);
if (curl_getinfo($ch, CURLINFO_HTTP_CODE) != 200) {
   trigger_error('Output request failed: ' . $response, E_USER_ERROR);
}
curl_close($ch);
```

If you did it right, the response code will be 200 (OK) or 201 (Created). If something is wrong, we always try to provide a clear error message in the request body. If the request succeeds, our servers will start rendering as soon as possible. Once the files are ready, your server will be notified as described in 4.3.

Note: if you exceeded your plan's monthly PDF limit and do not have credit in your account, the response code will be 402 (Payment Required).

4.3 WEBHOOKS

Once output files are ready, we'll send a POST request to an endpoint on your server. This goes for both preview files and PDF files. This POST is very simple:

Request		
Property	**Type**	**Description**
`apiKey`	String (*)	See 1.3
`url`	String (*)	The URL from which you can fetch JSON with more details.

The HTTP response of your system must have a status code in the 2xx range, and the body can be empty. Your server should follow the `url` in our request to fetch a JSON file with more details.

.NET code sample using `HttpClient` and `Newtonsoft.Json`:

```
// TODO validate: API key, URL (must start with https://output.printapi.io/)
var response = await httpClient.GetAsync(url);
var json = await response.Content.ReadAsStringAsync();
var output = JObject.Parse(json);
Console.WriteLine(output["snapshotId"]);
Console.WriteLine(output["type"]);
Console.WriteLine(output["fileUrls"]);
```

PHP code sample using `cURL`:

```
// TODO validate: API key, URL (must start with https://output.printapi.io/)
$json = file_get_contents($_POST['url']);
$output = json_decode($json, true);
print_r($output['snapshotId']);
print_r($output['type']);
print_r($output['fileUrls']);
```

Note that this webhook is fired for both preview and PDF files. Always check the `type` property.

As you can see in the given code samples, the response will be in this JSON format:

Response		
Property	**Type**	**Description**
`snapshotId`	String (32)	Same as the snapshotId pushed in 3.2. For looking up your cart/order item.
`type`	String	One of `"Pdf"` or `"Preview"`. See 4.1.
`fileUrls`	Object	Structure varies per `type`.

For `"Pdf"` output, the `fileUrls` object will be structured as follows:

PDF file URLs		
Property	**Type**	**Description**
`content`	String (*)	URL from which you can download the content PDF file. For most products, this is the only PDF file you'll need.
`cover`	String (*)	URL from which you can download the cover PDF file, if applicable. This PDF file is only present/needed for book-type products.

For `"Preview"` output, the `fileUrls` object will be structured as follows:

Preview file URLs		
Property	**Type**	**Description**
`content`	Array (String)	URLs from which you can download the content preview JPEG or PNG files. For most products, these are the only preview files you'll need. This array contains a file for every page of the project. Many products only support one page. In such cases, this array will contain only one element. Other products may support more pages (e.g. front/back), often a variable number (e.g. books). The first file URL in this array is page 1, the second is page 2, etc.
`cover`	Array (String)	URLs from which you can download the cover preview JPEG or PNG files, if applicable. This array is empty for non-book-type products. • The first file URL in this array is the front of the cover • The second file URL in this array is the back of the cover

5 CHECKLIST

Below you'll find a full list of all information we need. You can enter this info via the management panel of your Design Tool. (We can always change settings at a later time.)

5.1 ENDPOINT URLS TO PREPARE

URLs for the API endpoints in your system, built according to the specs in this document.

URL name	**Required?**	**Specified in chapter**
Log in user	Yes	2.3
Retrieve user ID	Yes	0
Push snapshot	Yes	3.2
Mark project as deleted	No	3.3
"Output file ready" webhook	No	4.3
Import album	No	6.3

5.2 STATIC URLS TO PREPARE

Links to website pages to which we may redirect the user.

URL name	**Required?**	**Description**
Home page	Yes	Destination of the app logo (top-left).
Cart page *	Yes	Destination of the cart redirect (see 3.2).
Registration page **	Yes	Destination of "Register" links.
Contact page	Yes	Destination of "Customer service" links.
Privacy policy	Yes	Destination of "Privacy policy" links.

* If you want to dynamically redirect the user (e.g. based on what item they just added to the cart), you can supply a token as described in 3.2. Example: `redirect_url = cart_url + token`

** We'll append a '*return URL*' before redirecting the user. Your server should redirect the user back to this URL after registration. Example: `redirect_url = register_url + current_url`

5.3 OTHER SETTINGS TO PREPARE

Miscellaneous settings that determine the behavior of your Design Tool.

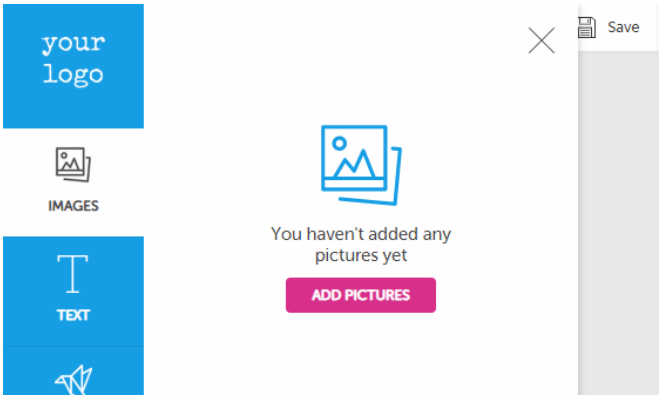
Setting name	**Required?**	**Description**
Application name	Yes	Displayed in the title bar
Currency symbol	Yes	Like € or $
Error e-mail recipient	Yes	May be notified if webhooks or other calls are failing.
Custom API key	No	See 1.3. If not supplied, we'll generate one for you.
Cookie domain	Yes	See 2.1. Usually: `.your-primary-domain.com`
Login cookie name	Yes *	See 2.1. Defaults to `auth`.

*Login cookie name is not needed if we enable the "user impersonation mode" described in 2.1, because our server will be sending and accepting all cookies indiscriminately anyway.

6 APPENDIX 1. ALBUM IMPORT API

6.1 OVERVIEW

The "images" tab in the Design Tool normally starts off empty. The user manually populates this tab by uploading images from their device, or by importing them from social media.



But maybe your service already hosts your users' images. In that case, you may want those images to be available in the Design Tool right away, without requiring the user to manually upload or import them. If so, you can implement the API specified in this chapter.

This is an optional feature. If you don't need this kind of functionality, you don't need to implement this API at all.

In summary, the way this API works is as follows:

- On startup, the Design Tool will call an endpoint on your server
- Your server answers with a list of images in our predefined JSON format
- That list of images is immediately imported into the project

6.2 ALBUM ID PARAMETER

To trigger the album import feature, you **must** pass a query parameter named `albumId` when linking to the Design Tool. For example:

```
https://editor.your-domain.com/?albumId=i5Lm3A6pzI
https://editor.your-domain.com/create/some-product?albumId=recent
https://editor.your-domain.com/create/some-product?albumId=2019
```

The value of this parameter will be included in the call to your album endpoint later (see 6.2). What you use as the value depends on your use case: it can be anything from a UUID, to a static string like "`recent`" or a dummy value. Note that the call to your album endpoint will also include the user identity.

6.3 Endpoint: POST – Import album

As described in 6.1, this call happens when the Design Tool starts up. The splash screen will disappear once the photos have been imported. Below are the fields we'll POST (formatted "application/x-www-form-urlencoded" as always):

Request		
Property	**Type**	**Description**
`apiKey`	String (*)	See 1.3
`albumId`	String (*)	See 6.2
+ Current login cookie		

Your server must respond with a JSON object containing up to 300 images:

```
{
    "images": [
        {
            "name": "foo.jpg",
            "type": "image/jpeg",
            "width": 3456,
            "height": 2304,
            "utcCreated": 1562661900080,
            "urls": {
                "small": "https://www.example.com/foo-small.jpg",
                "large": "https://www.example.com/foo-large.jpg",
                "full": "https://www.example.com/foo-full.jpg"
            }
        },
        {
            "name": "bar.png",
            "type": "image/png",
            "width": 3456,
            "height": 2304,
            "utcCreated": 1562662020315,
            "urls": {
                "small": "https://www.example.com/bar-small.png",
                "large": "https://www.example.com/bar-large.png",
                "full": "https://www.example.com/bar-full.png"
            }
        }
    ]
}
```

The details of the image object are as follows:

Image		
Property	**Type**	**Description**
`name`	String (*)	File name, used for sorting by alphabet.
`type`	String (*)	MIME type. One of "image/jpeg" or "image/png".
`width`	Number	The original width of the image in pixels.
`height`	Number	The original height of the image in pixels.
`utcCreated`	Number	Optional Unix timestamp (ms), used for sorting by date.
`urls.small`	String (*)	HTTPS URL of image file – Max 250px (longest side)
`urls.large`	String (*)	HTTPS URL of image file – Max 1250px (longest side)
`urls.full`	String (*)	HTTPS URL of image file – Original resolution (max 30 MB)

Note that your image URLs will be *directly* used by the Design Tool to load and display the images – at least at first. This means you are responsible for the following:

- Ensuring the URLs support *Cross-Origin Resource Sharing* (CORS) requests
- Ensuring the server responses are fast (target below 200ms TTFB)
- Ensuring the file sizes are reasonable (target below 20kb for `small`, below 300kb for `large`)
- Ensuring the image files are valid JPEG or PNG files (not corrupted)
- Ensuring long-term availability of the URLs with solid uptime guarantees

Using a CDN is therefore recommended. Our server typically downloads the images in the background, but this is not guaranteed and takes an indeterminate amount of time.

6.3.1 Error handling

If one of your image URLs fails, that image will appear in the Design Tool in an invalid state. It will be retried as the user navigates around the Design Tool or if they reload the page. If the failure happens when rendering a PDF or preview, you may be contacted.