Add all information to project files, organize in a way that serves the project and your memory the best:

oAuth

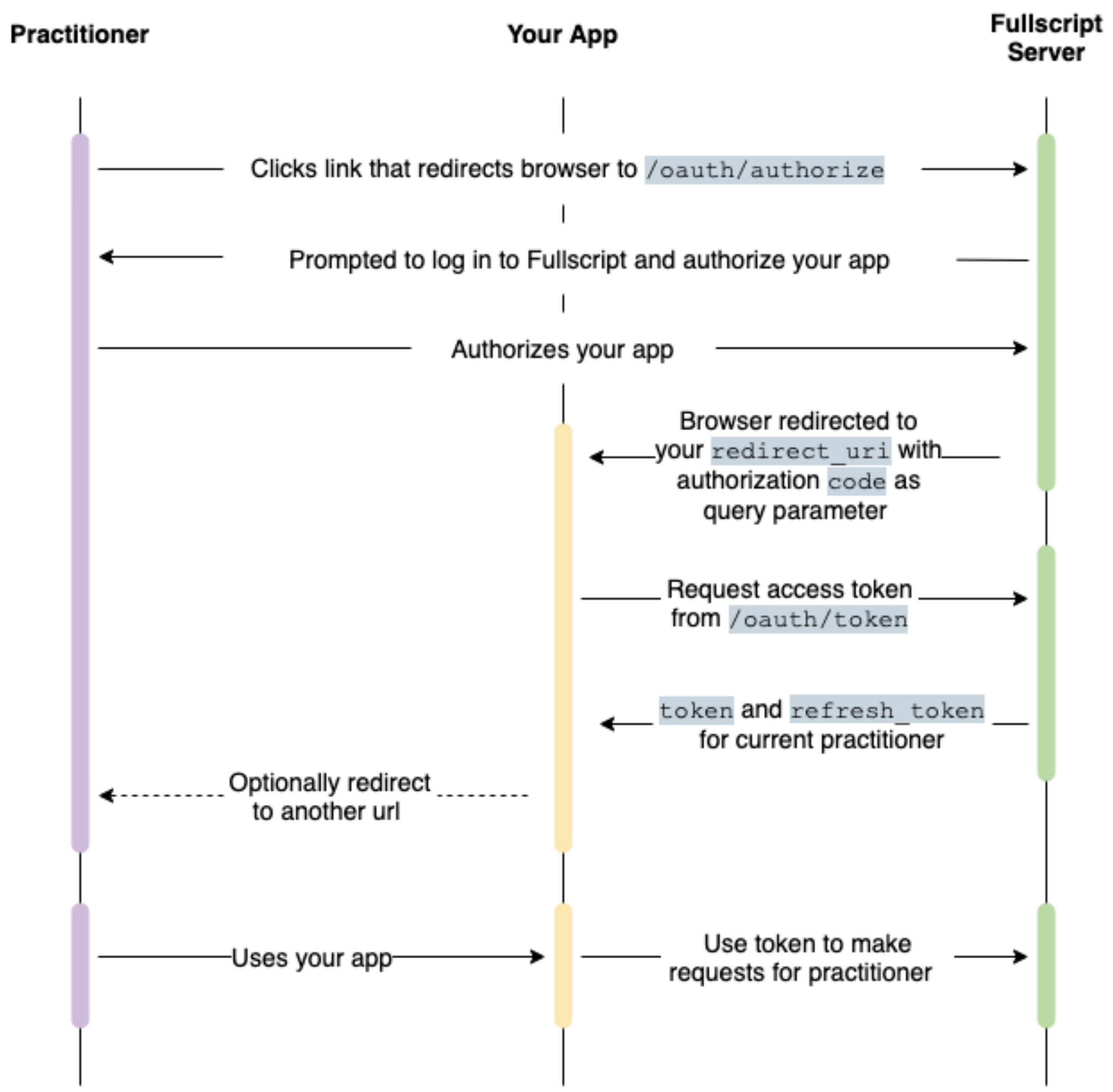
OAuth overview

When developing a Fullscript integration, your app’s Fullscript API interactions are done on behalf of Fullscript users. Until your app is authorized by at least one Fullscript user, there’s very little you can do with our APIs.

OAuth scheme

Fullscript uses the OAuth 2.0 protocol with role-based access control. So, one of the first things to set up in your app is the OAuth flow.

We support the standard authorization code flow grant type. In this scheme, you obtain an authorization code from our API endpoint by asking each user to authorize your app to access their Fullscript private data. Your app then exchanges this authorization code for a secure access token that is used to access all the other API endpoints on behalf of the authorizing user.


Practitioner
Your App
Fullscript Server
Clicks link that redirects browser to /oauth/authorize
Prompted to log in to Fullscript and authorize your app
Authorizes your app
Browser redirected to your redirect_uri with authorization code as query parameter
Request access token from /oauth/token
token and refresh_token for current practitioner
Optionally redirect to another url
Uses your app
Use token to make requests for practitioner

OAuth steps

Let’s take a closer look at the OAuth authorization code flow. It can be broken down into 3 steps.

1. Users grant access

Fullscript users (practitioners and office staff) are asked to grant your application access to their Fullscript account

In your app, add a place for users to trigger the authorization process to connect their Fullscript account. This can be a **Connect my Fullscript account** button in a special settings page, or it can be triggered via your app’s usual **add a Fullscript treatment plan** flow.

When the link is clicked by a user who hasn’t completed authorization or doesn’t have a valid access or refresh token, redirect them to our OAuth page.

See Request an auth code for implementation details.

Button assets

We have a pre-created button pack you can use. Or, use our logo files to create your own. Here are three of our most commonly used colors:

- grey (#36485C) ■
- green (#88B04B) 🟩
- coal (#2E3A47) ■

2. Redirect back to your app

As part of the OAuth setup, you'll give us a `redirect_uri`, which is an endpoint provided by your app. If the practitioner authorizes your application, we'll forward them to your `redirect_uri` with a **one-time-use authorization code** in the url query parameters.

You have **10 minutes** to exchange this authorization code for a secure access token (described in step 3, below) or the authorization code expires and you’ll need to start back at step 1.

Tip

Generally, app developers don't expost the `redirect_uri` page to users. They store the authorization code, then immediately redirect the user to another spot in their app.

3. Token exchange (the OAuth dance 🕺)

Behind the scenes, your app exchanges the temporary auth code for an **access token** and can then access the Fullscript API.

OAuth access tokens expire in **2 hours**. But they also come with a non-expiring refresh token so you can refresh an expired token when needed.

See Request an access token and Refresh an access token for implementation details

FYI

Applications created before August 26, 2021 use OAuth 2.0 with **clinic** level authorization instead of role-based access control. This guide outlines the differences (if any) for apps using clinic level authorization. Look for **FYI** blocks like this one.

Using oAuth:

Setup

To get started with OAuth you you need your **App**'s client ID (`client_id`) and client secret (`client_secret`). These identify to our server that your app's requests are coming from you.

Then, you'll need to set your app's OAuth scopes, which will be displayed to the user before they grant your app permission to their Fullscript data.

Access the API Dashboard

If you don't already have an **App** created with Fullscript, sign in to the Fullscript API Dashboard and click **Create Application** (or continue to sign up if you're brand new to us!). For initial development, target one of our sandbox servers (either US or Canadian).

You'll need an application name (currently used for your information only), and a redirect uri where we will send users once they have authorized your app's request to access their Fullscript data.

Tip
Both the application name and redirect uri can be edited later, so there's no need to agonize over those decisions now.

Client ID and secret

To find your **App**'s client ID and client secret, locate your **App** in the Fullscript API Dashboard and click **Show more details**. Your client ID is at the top of the page, and your secret key is masked below it. Hover your mouse over the **Reveal** button to see and **Copy** your secret key.

Tip
Your client secret is masked to remind you to keep this value a secret and handle it as you would handle any other important password.

Set app scopes

Scopes limit which pieces of the user's data you have access to, and what you can do with it. By default, your application is set up to request only some basic read information from the user's Fullscript data.

When a user is authorizing your app's access to Fullscript, they see the scopes you've set. It's up to them to decide if they want to authorize access to these scopes. We suggest you limit your application to the scopes that you need rather than ask for read/write access to everything.

Use the Fullscript API Dashboard to configure the scopes needed for each **App** you create.

Important
Adding scopes to your **App** instance in the Dashboard, doesn’t update your users’ previous authorization tokens with the new scopes. Attempts to use these tokens for activities associated with a new scope will return an `Unauthorized` error. Whenever you add scopes to your **App**, we suggest you obtain new tokens by prompting all your users to reauthorize.

Catalog scopes

`catalog:read`

Grants read-only access to the Fullscript catalog. This includes the brand, product, and variant endpoints. This is a default scope that all apps must request access to.

Clinic scopes

`clinic:read`

Grants read access to clinic’s account and practitioner information.

`clinic:write`

Grants access to a clinic’s account. Can add or edit users and can perform actions on behalf of the clinic's practitioners. When this scope is paired with `patients:read`, treatment plans can be created.

Patient scopes

`patients:read`

Grants access to read a patient’s data. This includes read access to any treatment plans made through the API. (Read access to treatment plans made through the Fullscript App, regardless of when they were created, requires the `patients:treatment_plan_history` scope.)

`patients:write`

Grants access to create and update a patient’s information.

`patients:treatment_plan_history`

Grants read access to a patient’s historical treatment plan data. Needed to access treatment plans created via the Fullscript App.

`patients:order_history`

Grants read access to a patient’s order history. Use this data to show patient adherence to the treatment plan.

Request an Auth code:

Request an auth code

Once you've retrieved your app's `client_id` and `client_secret`, and set your app's OAuth scopes, the next thing to do is have a place in your app that triggers the user authorization process. See our OAuth overview for some helpful tips on creating this interface in your UI.

Query

A request to `oauth/authorize` is different from requests to other parts of our API. Instead of making a programmatic call to one of our endpoints, `oauth/authorize` is done via a browser redirect. The user clicks your app's **Connect my Fullscript acccount** or **Add a Fullscript treatment plan** button and you redirect the user's browser to the appropriate Fullscript authorize url.

Sandbox applications

Use the appropriate (US or Canada) OAuth authorization url below for apps under development.

```
# 🇺🇸 US Sandbox (testing)
https://us-snd.fullscript.io/oauth/authorize?
client_id=YOUR_CLIENT_ID&redirect_uri=YOUR_APPLICATIONS
_REDIRECT_URI&response_type=code
# 🇨🇦 Canada Sandbox (testing)
https://ca-snd.fullscript.io/oauth/authorize?
client_id=YOUR_CLIENT_ID&redirect_uri=YOUR_APPLICATIONS
_REDIRECT_URI&response_type=code
```

Production applications

When you make your application live, don't forget to update the authorization URLs to target our production auth servers.

```
# 🇺🇸 US Production
https://api-us.fullscript.io/oauth/authorize?
client_id=YOUR_CLIENT_ID&redirect_uri=YOUR_APPLICATIONS
_REDIRECT_URI&response_type=code
# 🇨🇦 Canada Production
https://api-ca.fullscript.io/oauth/authorize?
client_id=YOUR_CLIENT_ID&redirect_uri=YOUR_APPLICATIONS
_REDIRECT_URI&response_type=code
```

Example request

As shown in the query definition above, you need your **App**'s `client_id`, retreived from the Fullscript API Dashboard in our setup step. You must also provide the exact same `redirect_uri` you configured on the Dashboard. Your request will return `404` if there's any mismatch.

For example:

```
https://api-us-snd.fullscript.io/api/oauth/authorize
  ?
client_id=83x92x21xx29x643xx9954x8x2xx651xxx87x66521x08
x9x2x408x26x801x394
  &redirect_uri=https://example.com/redirect
  &response_type=code
```

Tip

There's no need to add scopes to the request; any scopes added this way are ignored. Scopes you configured via the Fullscript API Dashboard are automatically applied to your request.

Optional state

You can optionally pass in a `state` parameter to the authorization code request. We don't process the state parameter, but instead, pass it back to you in the next step.

You can use the state paramater to do things like remember which page or part of a process your user was in before they were redirected to the Fullscript OAuth process.

If you want to keep track of more complex things, like the fact that the user was viewing a particular patient page, you can encode that data, then pass the encoded string in the `state` param:

```
state="some-encoded-state-data-that-my-app-needs"
```

To include the state option, just append it to your call:

```
https://api-us-snd.fullscript.io/api/oauth/authorize
  ?
client_id=83x92x21xx29x643xx9954x8x2xx651xxx87x66521x08
x9x2x408x26x801x394
  &redirect_uri=https://example.com/redirect
  &response_type=code
  &state=some-state-data-that-my-app-needs
```

User interaction

At this point, it's out of your hands for a few moments.

When redirected to our OAuth flow, users who aren’t already signed in to Fullscript are prompted to sign in or sign up for a Fullscript account. Once signed in, they are shown a detailed list of the access scopes your application is requesting, and they are asked to authorize your application.

The user can either cancel or authorize your application to access their Fullscript data.

Response (sort of)

Since this isn't your typical API call, it's not a typical response either.

Rather than having a response code returned to your application, users who authorize your app are redirected back to your `redirect_uri`. The query parameters returned include your new (temporary) authorization `code` for this user.

```
https://www.example.com/redirect
  ?
code=476x028x13xxxx52xx4x2xx87x472x735xxx893570x89xx48x
9496xxx2418052
```

If you passed in a state parameter, it's passed back to you:

```
https://www.example.com/redirect
  ?
code=476x028x13xxxx52xx4x2xx87x472x735xxx893570x89xx48x
9496xxx2418052
  &state=some-state-data-that-my-app-needs
```

Use the provided authorization `code` to call our OAuth token endpoint and request an access token to use on behalf of your user.

Important
The authorization `code` has a **10 minute lifetime**. You must use it to request an OAuth access token within 10 minutes, or you'll need to re-request user authorization.

Setup
Request an access token

Request an access token

Now it's up to you to do the OAuth dance 🕺! The temporary authorization code you received must be exchanged for an OAuth **access token**.

To do so, you make an API request with your OAuth application credentials and the authorization code. If successful, our server generates an OAuth token and includes it in the response.

Select an API server

During development use:

- US: `https://api-us-snd.fullscript.io/api` 🇺🇸
- CA: `https://api-ca-snd.fullscript.io/api` 🇨🇦

And once your app is live (available after your development app is approved by our team), switch to:

- US: `https://api-us.fullscript.io/api` 🇺🇸 🌐
- CA: `https://api-ca.fullscript.io/api` 🇨🇦 🌐

Endpoint

Your HTTP request for an access token goes to our `api/oauth/token` endpoint on the correct sandbox or production server (remember to set US vs Canada).

For example, a test query for an app targeting our US servers uses this endpoint:

```
POST https://api-us-snd.fullscript.io/api/oauth/token
```

Example request

The same endpoint can both create a new access token or refresh an existing one. To let the server know your call is to trade an authorization `code` for a **new** access token, set the `grant_type` to `authorization_code` and set `code` to the value received when the user authorized your app.

This is also the first time you need to provide the `client_secret` you copied from the Fullscript API Dashboard during OAuth setup.

```
curl -X "POST" "https://api-us-snd.fullscript.io/api/
oauth/token" \
  -H 'Content-Type: application/json' \
  -d $'{
  "grant_type": "authorization_code",
  "client_id":
"83x92x21xx29x643xx9954x8x2xx651xxx87x66521x08x9x2x408x
26x801x394",
  "client_secret":
"5305x72xxx6xx7xx4848xxxxx040x4xx4xx965xx566x662xxxxx6x
7xxx5x730x",
  "code":
"x8x9xx06x9138x41x0x4xx3xxx726x0x2749xxx798x1x7x3x46x90
2x4068664x",
  "redirect_uri": "https://example.com/redirect"
}'
```

Example response

Assuming you made the call within 10 minutes of receiving the authorization code, our servers respond with the user's OAuth access token. You can think of an `access_token` as a user's identity and password combined into a single field.

```
{
  "oauth": {
    "access_token":
"96x5x67x09168xx64x4x8xx5xx541xxxxx38x10071xx1xxx24x66x
0xxxxxxx74",
    "token_type": "Bearer",
    "expires_in": 7200,
    "refresh_token":
"xxxx4x98xxxx7x03xxxxxx73x95x2514x0xx2x9xx15882xx794749
6376x343x9",
    "scope": "catalog:read",
    "created_at": "2021-06-16T14:57:21.000Z",
    "resource_owner": {
      "id": "xx7x357x-9x36-xxxx-x553-7x3xx398xxx",
      "type": "Practitioner"
    }
  }
}
```

The OAuth access token has a 2 hour lifetime, which you can calculate from the provided `created_at` and `expires_in` values. But they also come with a non-expiring `refresh_token` so you can refresh an expired token when needed.

Store the `access_token`, `refresh_token`, the user’s Fullscript `id` and user `type` (also returned with the token) alongside other private user profile info in your app.

Use the access token

Once you've successfully received and stored the OAuth token, use it to interact with the Fullscript API on behalf of that authorizing user.

Tip

Each user (practitioner or staff member) needs their own unique OAuth token so Fullscript knows which user is interacting with the clinic data.

To make a request on behalf of a user, just include their token in the header of your request, like so:

```
{ "Authorization" => "Bearer
96x5x67x09168xx64x4x8xx5xx541xxxxx38x10071xx1xxx24x66x0
xxxxxxx74" }
```

And that's it! You've done the OAuth dance, and you can now make calls on behalf of your user.

Just one more thing. 😉 Remember we said that the token expires in two hours. Avoid making the user go through the whole authorization process each time they use the API, and instead, use our refresh token process to simply refresh their expired OAuth token.

FYI
Applications created before August 26, 2021 use OAuth 2.0 with **clinic** level authorization instead of role-based access control. For those apps, there's no requirement to create and store an OAuth token per user, but it's still good practice to do so. If your app allows users to connect their own Fullscript clinic accounts, you may already be doing this.

Refresh an access token

Once your app has a users's OAuth token, you should ensure it's unexpired before using it.

Check validity

As you recall, the access code response includes both a timestamp when it was created, and an expiry time, in seconds. So you can know in advance of using an access code if you might need to refresh it first.

Endpoint

Refreshing an access token is easy and uses the same endpoint you used to create it: `api/oauth/token`. (Remember to specify the same sandbox or production server used when creating the token.)

For example, a test query for an app targeting our US servers uses this endpoint:

```
POST https://api-us-snd.fullscript.io/api/oauth/token
```

Example request

When refreshing a token, specify `refresh_token` as the `grant_type` and instead of including an authorization `code`, include the `refresh_token` you received with the access token.

```
curl -X "POST" "https://api-us-snd.fullscript.io/api/
oauth/token" \
  -H 'Content-Type: application/json' \
  -d $'{
  "grant_type": "refresh_token",
  "client_id":
"83x92x21xx29x643xx9954x8x2xx651xxx87x66521x08x9x2x408x
26x801x394",
  "client_secret":
"5305x72xxx6xx7xx4848xxxxx040x4xx4xx965xx566x662xxxxx6x
7xxx5x730x",
  "refresh_token":
"xxxx4x98xxxx7x03xxxxxx73x95x2514x0xx2x9xx15882xx794749
6376x343x9",
  "redirect_uri": "https://example.com/redirect"
}'
```

Example response

As long as

- you're requesting a refresh token from the same server that gave you the access token; and
- the user hasn't revoked their authorization of your application

you'll get back a response that includes a new OAuth access token.

The response parameters are the same as you receive when getting the user's first access token, but will have an updated `created_at` timestamp and a new `refresh_token`.

```
200 OK
{
  "oauth": {
    "access_token":
"96x5x67x09168xx64x4x8xx5xx541xxxxx38x10071xx1xxx24x66x
0xxxxxxx74",
    "token_type": "Bearer",
    "expires_in": 7200,
    "refresh_token":
"xxxx4x98xxxx7x03xxxxxx73x95x2514x0xx2x9xx15882xx794749
6376x343x9",
    "scope": "catalog:read",
    "created_at": "2021-06-16T14:57:21.000Z",
    "resource_owner": {
      "id": "xx7x357x-9x36-xxxx-x553-7x3xx398xxx",
      "type": "Practitioner"
    }
  }
}
```

The `refresh_token` you used to make this call will not expire until you use the newly obtained `access_token`. If an error occurred while processing the response on your end, you can retry the request with the old `refresh_token`. After the first use of the new `access_token`, only the new `refresh_token` will be valid.

Revoke an access token

Since users granted your application access to their data via a process in your application, it's nice to also provide them a way to revoke access if needed. You'll find this comes in handy for testing, too.

In our experience, most integrations call this **Disconnect my Fullscript account** to be a natural opposite from **Connect to Fullscript** or **Connect my Fullscript account**.

FYI

Practitioners and staff members can also revoke your app's access through the Fullscript Web App.

Endpoint

To revoke an access token use the `api/oauth/revoke` endpoint. The token doesn't need to be refreshed prior to revoking it, and you don't need to provide a bearer token for this call. (Remember to specify the same sandbox or production server used when creating the token.)

For example, a test query for an app targeting our US servers uses this endpoint:

```
POST https://api-us-snd.fullscript.io/api/oauth/revoke
```

Example request

To revoke a token, you need only provide your `client_id`, `client_secret`, and the `token` you want to revoke.

```
curl "https://api-us-snd.fullscript.io/api/oauth/revoke" \
  -H 'Content-Type: application/json' \
  -d $'{
  "client_id":
"83x92x21xx29x643xx9954x8x2xx651xxx87x66521x08x9x2x408x26x801x394",
  "client_secret":
"5305x72xxx6xx7xx4848xxxxx040x4xx4xx965xx566x662xxxxx6x7xxx5x730x",
  "token": "xx7xxxxx7xxxx3xxxx5xxx9xx4xxxxxx1xxxx3x9xxx"
}'
```

Example response

This call returns an empty body.

```
200 OK
{
}
```

What is Fullscript Embed?

Previously referred to as Fullscript.js, Fullscript Embed is our latest embeddable experience.

Fullscript Embed is a pre-built client-side library that integrates with the Fullscript API, removing the need to build a custom frontend or logic for Fullscript actions. Instead, you simply load Fullscript Embed, which provides convenient methods to seamlessly integrate the Fullscript experience into your application.

For example, you can embed the Fullscript experience directly within a page or as a modal dialog. This allows practitioners to prescribe from Fullscript's supplement catalog without leaving your app.

In addition, **Fullscript Embed** alerts you of events that you can capture in your application to support comprehensive patient care. For example, you can listen for the **"treatment plan activated"** event, which triggers when a practitioner sends a patient's treatment plan. You'll receive the full treatment plan details, allowing you to store and track practitioner and patient actions as needed.

Fullscript Embed integration overview

We've put together a list of links and design information that you can review before you start coding.

Once you're through that, we've broken the integration into 5 steps:

1. Setup
2. OAuth for user authentication

3. Add an embeddable component
4. Handle events
5. Go live! 🎉

You’ll also find useful information in the Error Handling section of this document.

Important links

Here are the links you're most likely to use as you develop your Fullscript integration.

Fullscript API dashboard

Your primary destination to register App instances with our API, get OAuth setup details, register for webhooks, and more.

- Go to the Fullscript API Dashboard

API reference docs

Find a detailed listing of all our API endpoints. Includes examples, inputs, outputs, and data object definitions.

- Technical reference

Development sandboxes

API endpoints to use during development. Tests your integration without affecting live patient records. You can use the Create Application function in the Fullscript API Dashboard to create multiple test Apps per target (US and Canada). This can be helpful for testing and debugging different builds of your app.

- US: `https://api-us-snd.fullscript.io/api` 🇺🇸
- CA: `https://api-ca-snd.fullscript.io/api` 🇨🇦

Production servers

Live production API endpoints. Only for use with production software. You'll be able to create production applications once your sandbox integrations have been approved by our team.

- US: `https://api-us.fullscript.io/api` 🇺🇸 🌐
- CA: `https://api-ca.fullscript.io/api` 🇨🇦 🌐

Important links

Here are the links you're most likely to use as you develop your Fullscript integration.

Fullscript API dashboard

Your primary destination to register App instances with our API, get OAuth setup details, register for webhooks, and more.

- Go to the Fullscript API Dashboard

API reference docs

Find a detailed listing of all our API endpoints. Includes examples, inputs, outputs, and data object definitions.

- Technical reference

Development sandboxes

API endpoints to use during development. Tests your integration without affecting live patient records. You can use the Create Application function in the Fullscript API Dashboard to create multiple test Apps per target (US and Canada). This can be helpful for testing and debugging different builds of your app.

- US: `https://api-us-snd.fullscript.io/api` 🇺🇸
- CA: `https://api-ca-snd.fullscript.io/api` 🇨🇦

Production servers

Live production API endpoints. Only for use with production software. You'll be able to create production applications once your sandbox integrations have been approved by our team.

- US: `https://api-us.fullscript.io/api` 🇺🇸 🌐
- CA: `https://api-ca.fullscript.io/api` 🇨🇦 🌐

Development sandbox

Development happens against our sandbox servers. Before you go live, we have a review process that gives you access to our production servers.

Our sandbox environments are nearly identical to production. As you read these docs, you'll notice we highlight any differences you need to be aware of.

To get you started, here are the top three sandbox vs production considerations:

1. Sandbox and production servers have different target URLs to use in your API calls. See Important links for details.

   FYI
   Regulations and our product catalogs vary slightly between the USA and Canada, so we have dedicated servers (sandbox and production) for each. If your software (or platform) operates in both Canada and the USA, you need two copies of your **App**. One for each target.

2. Sandbox Apps have OAuth keys and secret strings that are unique and will be different in your production App.
3. For obvious reasons, our sandbox environment doesn’t process actual patient supplement orders. However, you can place test orders using Stripe’s test credit card:
   ```
   4242 4242 4242 4242 (with any future expiry date)
   ```
4.
5.

Tip

To simplify your switch from development to production, we suggest storing server-specific info, such as the base target URL and your OAuth secret keys, in environment variables or using another easy to swap technique.

At the end of this guide, you’ll find a Go Live checklist to make sure your launch day goes smoothly.

Design decisions

There are some common design questions that will come up as you work through your Fullscript integration. We’ve tackled these below to give you a starting point for your discussions.

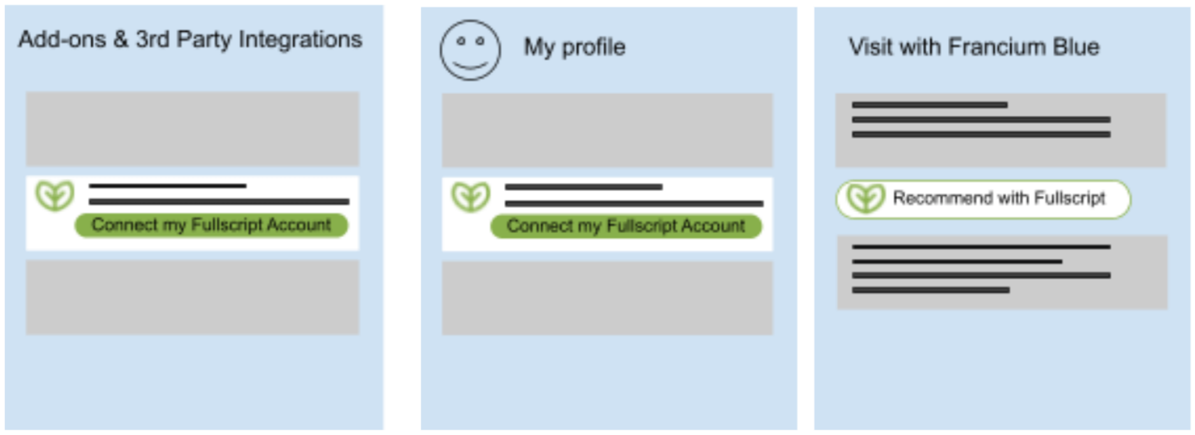
How will users discover the Fullscript integration?

Fullscript APIs use role-based OAuth, which means each user authorizes their own Fullscript account access - even if all the accounts are part of the same Fullscript clinic.

Most of our API calls require the user’s OAuth Bearer token to identify who you’re making the call on behalf of. Getting that authorization is one of the first design issues to address.

The three most common ways our partners make the integration available to their users are:

- on a third party plugins page
- on the user's profile page
- by triggering the OAuth flow the first time the user interacts with the integration



Add-ons & 3rd Party Integrations
Connect my Fullscript Account
My profile
Connect my Fullscript Account
Visit with Francium Blue
Recommend with Fullscript

If your app already has a section for integrations or third party plug-ins, your users know where to go looking for add-ons. Add a **"Connect my Fullscript account"** button that triggers the user's OAuth flow. On patient interaction pages, show or hide the **"Recommend with Fullscript"** button depending whether you have an auth token for the current user.

Alternatively (or even in parallel) you can choose to always show the **"Recommend with Fullscript"** button on the page(s) where it's relevant. If the user hasn't already completed the OAuth process, simply route them there before continuing with the normal button operation. This is a great solution for systems that don't have a dedicated integrations page, or who want to make the Fullscript integration very visible.

FYI

There is no API available to "toggle" the integration's availability for all users. If your design requires that a particular user enables (or disables) the ability to use the Fullscript integration, you'll need to create your own toggle. Each user will still need to proceed with their own OAuth authorization.

How will users revoke or remove the Fullscript integration?

As part of your integration, make sure you include a place for users to revoke their OAuth authorization. Not only is this the inverse to letting users authorize access, it's also super helpful for testing your integration. As a developer, you will no doubt use this feature more than any real user!

Generally, our partners put this feature on their third party integrations page or in the user's profile.

We also suggest you add:

- a master "revoke all Fullscript user authorization" button that's accessible to developers and admin users
- a way for admins to remove a selected user's authorization for employee lifecycle management

Tip
No bearer token is needed to revoke a token. You only need your app's client ID, client secret, and the token to revoke.

FYI
Practitioners can also revoke authorization from the Fullscript Web App.

How much to cross-index between systems?

To search for a patient or a practitioner via the Fullscript API, you often provide the user's email address. This gives you the fastest match because email address is a unique value in our system.

However, particularly in the case of patients, we find that some of our users share an email address. For example, a patient's guardian may use their email address for orders for their charge and for themselves - meaning that two or more people share the same Fullscript patient account.

This can make it harder for your app to match a patient in your system with one in ours. To simplify, certain resources in our system have an available `metadata` field where you can store your system's unique identifier. Then you can use the metadata endpoint to search our system for those resources later.

The following objects accept metadata when being created or updated:

- `patient`
- `practitioner`
- `staff`
- `treatment_plan`

Additionally, some of our patient-related REST API calls require the Fullscript patient id (which you can find with search for patients. If this becomes too taxing, you may choose to store our patient id with your records so you can skip the search step the next time.

When designing your system, you can choose none, one, or both of these cross-indexing options.

Tip
Remember that in the Fullscript account model, patients are owned by a clinic. If your app supports multiple clinics, a single patient record in your application could be tied to multiple Fullscript clinics. Each of their Fullscript accounts has a unique email address.

Will you store or fetch Fullscript data (or

both!)?

Depending on how much Fullscript data you show users, you may wish to store Fullscript data locally rather than use the API to fetch it every time. There may also be key user interaction pages where you start with what you have, but always want to fetch the latest data.

For example, let's consider a patient history page where you plan to display previous encounters and any resulting Fullscript treatment plans. You could fetch this data every time you need it, waiting to fill those sections of the page as the data is retrieved. Or, you could show the data you have on hand, then verify with Fullscript if anything new or changed since you last fetched it, and update if necessary.

Similarly, if you're integrating with Fullscript Redirect, you can update the patient page with the latest data after the practitioners saves the new treatment plan. For more details, see our Fullscript Redirect guide to updating the patient encounter page.

There are three ways to get Fullscript data:

- fetch the current data via the applicable API endpoints - this provides your app information as-needed
- register for webhooks and receive updates to data of interest to you - some partners use this for on-demand updates only, others use it to store event data for later
- set up a regular job to query our events endpoint - used to store data for later

FYI

Webhooks are triggered for actions your practitioner takes through the Fullscript Redirect URL, but **not** for anything you do specifically through the API. For example, if you use the `patients` endpoint to update the spelling of a Fullscript patient's name, we won't alert you of this since we're pretty sure you already know!

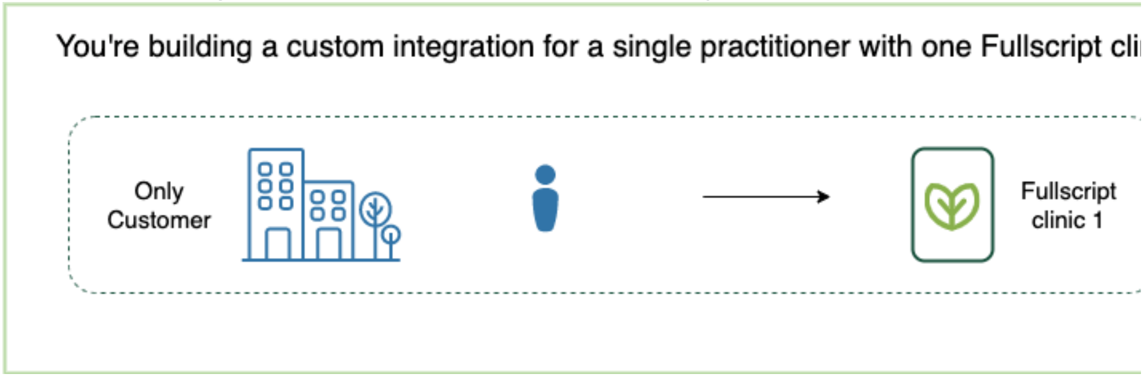
How will you support multiple practitioners?

When designing your interactions with Fullscript, it's helpful to understand our clinic and user account structure.

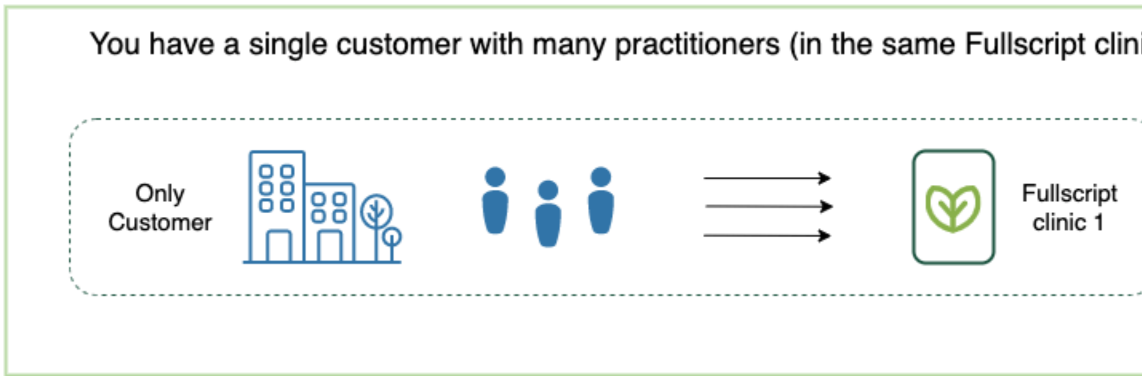
Fullscript divides its users into what we call **clinics**. A clinic has an **owner**, one or more **practitioners** (including the owner), **staff members** (also called **clerks**) who enter data on behalf of practitioners, and a set of **patients** with related **treatment plans**. Each practitioner and staff member has their own Fullscript account, with login tied to their email address.

We find there are four common use cases when it comes to integrating with Fullscript.

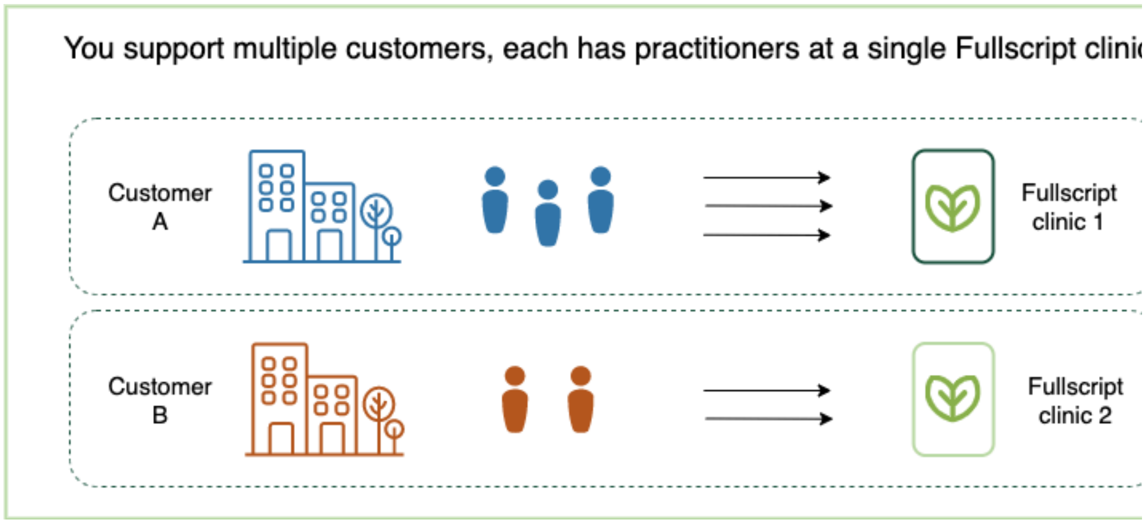
1. You're building an integration for one customer. Your customer is the owner of a single Fullscript clinic and is the only practitioner.


You're building a custom integration for a single practitioner with one Fullscript cli
Only Customer
Fullscript clinic 1

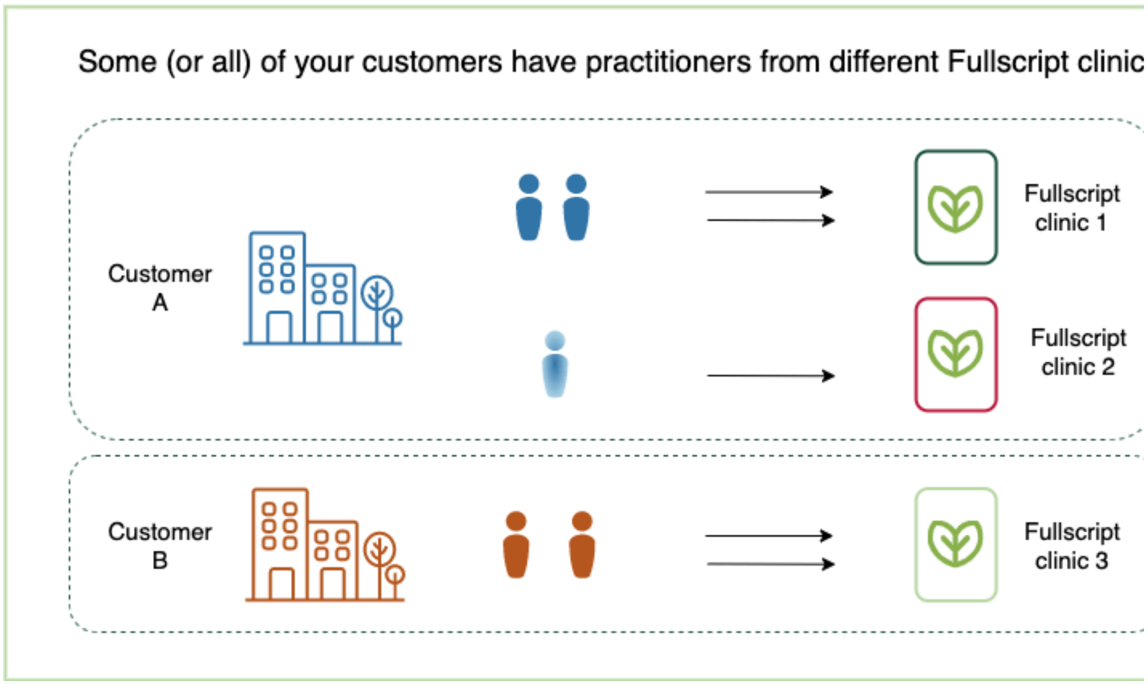
2. You're building an integration for one customer who owns a single Fullscript clinic. This clinic has multiple practitioners.


You have a single customer with many practitioners (in the same Fullscript clini
Only Customer
Fullscript clinic 1

3 You're building a system to be used by multiple customers. Your customers' Fullscript accounts are each in their own (single) Fullscript clinic.


You support multiple customers, each has practitioners at a single Fullscript clini
Customer A
Fullscript clinic 1
Customer B
Fullscript clinic 2

4 Your system can be used by one or more customers. Each customer may have one or more Fullscript clinics.


Some (or all) of your customers have practitioners from different Fullscript clinic
Customer A
Fullscript clinic 1
Fullscript clinic 2
Customer B
Fullscript clinic 3

It’s useful to note that Fullscript clinics and accounts are free to create. Each account simply needs a unique email address. Given it’s free to make multiple accounts, some practitioners have accounts at different Fullscript clinics.

For example, a practitioner may work for two different organizations and have an account with the Fullscript clinic associated with each of those companies. Or, a practitioner may have one Fullscript clinic configured for use with a philanthropic practice, and another for their main practice.

Tip

If your app supports multiple Fullscript users (regardless of whether that’s through a single clinic or multiple clinics), there’s no requirement that the Fullscript accounts are pre-created. Your customer’s Fullscript account owner can pre-invite their users through the Fullscript Web App, but users can also simply sign up during your app’s OAuth process.

Supporting a single Fullscript clinic

This is the simplest integration path. Regardless of which of our first two use cases you’re building for, you’re only supporting a single Fullscript clinic. The clinic has one or more

Fullscript user accounts (practitioners and staff members).

- With only one clinic to manage, there's no concern for which clinic a practitioner, patient, or event belongs to.

Supporting multiple Fullscript clinics

This is the case when your app has multiple unrelated sets of customers. It matches with our last two use cases above.

- Your app may need to keep track of which clinic a practitioner or patient belongs to. For example, you may have a reason to keep a list of practitioners per Fullscript clinic. To know which clinic a practitioner or staff member belongs to - use the retrieve a clinic endpoint with their OAuth token. Store the `clinic_id` with the user's data so you know which clinic is being accessed for future calls made with the user's OAuth token.
- If your app signs up for Fullscript webhooks or uses our events endpoint, your app may need to know which clinic the activity occurred with to know which set of data to update. Find this in the payload's `clinic_id` parameter.

Setup

If you don't already have an App created with Fullscript, sign in to the Fullscript API Dashboard and click **Create Application** (or continue to sign up if you're brand new to us!). For initial development, target one of our sandbox servers (either US or Canadian).

You'll need an application name (currently used for your information only), and a redirect uri where we will send users once they have authorized your app's request to access their Fullscript data.

Tip
Both the application name and redirect uri can be edited later, so there's no need to agonize over those decisions now.

Configure scopes

The Fullscript Embed integration process needs the following OAuth scopes:

- `Clinic:read`
- `Clinic:write`
- `Patients:write`
- `Patients:treatment_plan_history`

Configure your app to use these via the Fullscript API Dashboard's OAuth scopes settings for your App. Don’t forget to save your changes.

Create test users

Create one or more test practitioners by clicking the Create Test Account button on the right side of your App’s page in the Fullscript API Dashboard. For now, choose Skip when asked to upload certifications for your new test users. Stop when you get to the authorization screen for your new user. It requires you to have your redirect endpoint available. We’ll get to that in a few more steps.

FYI

When you create test practitioners as described above, they are automatically approved and there’s no need to upload fake certifications for your test users.

You’ll also need at least one test patient for your integration. Use the **Patient Sign Up** link from your practitioner’s welcome email, or log in to the Fullscript App as the practitioner, then select **Patients > New Patient**.

Here are the practitioner login links for the Fullscript sandbox Apps:

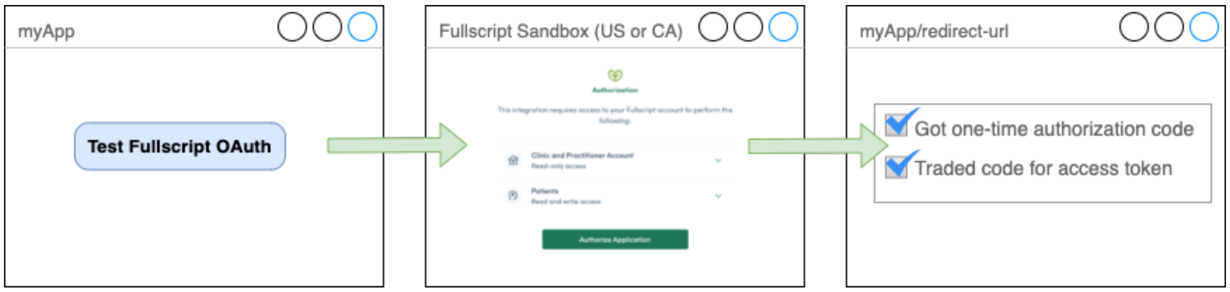
- US Sandbox: https://us-snd.fullscript.io/login
- Canadian Sandbox: https://ca-snd.fullscript.io/login

OAuth and backend work

By default, apps don’t have access to most functionality in the Fullscript API. Users own the data they’ve entered into Fullscript, and it’s up to them to grant your app access to that data.

Fullscript uses the OAuth 2.0 “authorization code flow” for API access, including access via Fullscript Embed. Follow our OAuth tutorial to add and configure OAuth.

Create a simple test flow something like this to make sure that your app has OAuth working and your redirect endpoint properly enabled:


myApp
Test Fullscript OAuth
Fullscript Sandbox (US or CA)
Authorization
This integration requires access to your Fullscript account to perform the following:
Clinic and Practitioner Account
Read-only access
Patients
Read and write access
Authorize Application
myApp/redirect-url
Got one-time authorization code
Traded code for access token

Tip
If your application supports more than one customer or more than one Fullscript clinic, store the user’s `clinic_id` (found by retrieving the clinic info, using their OAuth token as the Bearer token) with the practitioner’s user data. Begin grouping Fullscript data together by the `clinic_id`.

Important
Before continuing, run through the OAuth process with one of your test practitioners (created earlier) to ensure your app can retrieve the user’s `access_token`.

Codify session grants

Fullscript Embed has an extra authorization step that isn't present in our other integration methods. The extra step is to request a user **session grant** object that combines several attributes of a Fullscript user into a time-limited, single-use `secret_token` that your app shares for authentication when the embeddable feature is loaded.

In day to day operation, your app will retrieve a user's session grant right before initializing your chosen feature (because the session grant's `secret_token` must be used within 120 seconds of retrieving it). However, since we're already working with other REST API calls, this is a good time to get the prep work done.

Tip
The `secret_token` login context is agnostic of active sessions on the browser. Meaning if a different user is logged into Fullscript in another tab, their session will not change and the user active in the embedded section of your page will be the one specified using this api request.

Add a backend function to your app to get a user's session grant and related secret token. At a minimum, your session grant function takes in the user's id (or has another way to get the active user) and returns a session grant for that user. Later in this guide, you'll call this session grant function from your app's frontend.

Use the Fullscript `session_grants` API endpoint to get the session grant. Full details for this endpoint are available in the reference section.

Here's what a call to the `session_grants` endpoint looks like with curl (replace `xxxxxxxxxxxxxxxxxxxx` with the user's OAuth access token):

```
curl -X "POST" "https://api-us-snd.fullscript.io/api/
clinic/embeddable/session_grants" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Important
Build out your OAuth test page to include a session grant request. Test that you’re able to get a `secret_token` for your test practitioner.

Okay! 🎉 We completed all the backend work. It’s time to move to your app’s frontend.

Add embeddable component

Install Fullscript Embed

Moving into the frontend work, let’s get Fullscript Embed installed. We offer two functionally equivalent solutions, compiled from the same source:

- a UMD (Universal Module Definition)
- an NPM package (Node Package Manager)

The UMD version is a single JavaScript file packaged for direct consumption by a web browser. The NPM package can be used by any application with a bundler such as Parcel, Webpack etc.

Tip
The NPM package contains TypeScript information that enables autocomplete and suggestions in your IDE. For ease of use, we recommend accessing the Fullscript Embed package through the NPM.
Unless otherwise specified, all instructions work for both the UMD Build and NPM package.

FYI
Our Fullscript Embed package is open source. You can view its release history (including change log) and code in the GitHub repo.

UMD Build

To use the **UMD build**, include it as a script tag in any page of your app where you plan to host a Fullscript Embed feature:

```
<script src="https://public-assets.fullscript.com/fullscript.js/
3.0.1/fullscript-js.umd.min.js"></script>
```

Tip
Make sure the version in the string above matches the version of Fullscript Embed you want to use (usually the latest version!). You can find the available versions here.

NPM Package

To use the **NPM package**, install it with:

```
cd your-application-folder
npm install --save @fullscript/fullscript-js
```

Or if you are using yarn:

```
cd your-application-folder
yarn add @fullscript/fullscript-js
```

Load Fullscript Embed

In your app's frontend code, import or require Fullscript Embed wherever you need.

```
import { Fullscript } from "@fullscript/fullscript-js";
```

Or if using CommonJS style imports:

```
const Fullscript = require("@fullscript/fullscript-js").Fullscript;
```

Get your app's public key

When working with Fullscript Embed, your app needs to identify itself to the Fullscript servers. There are two parts to this identification:

- User identification: OAuth access tokens + session grants (configured earlier)
- App identification: Fullscript Embed public key

Each app you configure in the Fullscript API Dashboard gets assigned a unique Fullscript Embed public key. The key is specific to the app **and** the target Fullscript environment the app works with. For example, the public key for an app configured to use the **US Sandbox** won't work for a copy of your app that targets the **US Production** environment.

To get your public key, go to your app in the Fullscript API Dashboard and select the **Fullscript Embed settings tab**. There are three things to do on this page.

1. Verify which Fullscript environment your app is targetting. At the top of the page, look for the flag icon and text next to it to identify one of:
    - US Sandbox
    - CA Sandbox
    - US Production
    - CA Production
2. Click the **pencil icon** next to **Origin URI** and add the URLs that your integration will be loaded from. For development, this might just be `http://localhost/*`, but you can add a space-separated list of any URLs needed. URLs added can be full endpoints or just domains, but they must include the protocol (`http` can be used in sandbox, `https` must be used for production).

    FYI
    Fullscript protects your integration by only allowing Fullscript Embed to load on pages that come from specific domains. **Origin URI** is a whitelist of domains that can use Fullscript Embed with your app's credentials.

3. Hover your mouse over the Reveal button and copy the Fullscript Embed public key for the current target environment.

Initialize Fullscript Embed

Return to your app's code and initialize Fullscript Embed with the value you just copied as your public key.

```
const fullscriptClient = Fullscript({
  publicKey: "xxxxx",
  env: "us|ca|us-snd|ca-snd",
});
```

Set the env attribute to match the Fullscript environment your app targets. During development, use one of our two **sandbox** environments.

- us - targets **US Production**

- ca - targets **CA Production**
- us-snd - targets **US Sandbox**
- ca-snd - targets **CA Sandbox**

Tip

The public key is specific to the configured app and the Fullscript environment it's targeting. When you go live, you'll configure a new app in the Fullscript API Dashboard, and you'll need to update both the `env` and the `publicKey` in the initialization code.

Add Platform feature

Now that you have a client, you can enable the platform feature on your page to access the full Fullscript experience, including managing treatment plans and patients, browsing the product catalog, and more.

Syntax:

```
const platformFeature = fullscriptClient.create('platform',
<options>);
```

As you know, Fullscript Embed requires authorization from your users. Your app should initially prompt users for this authorization via OAuth (triggering the backend code you created earlier). On subsequent times, you can directly show them the feature.

Your app must tell Fullscript who is performing the task, and for which patient. Since the current user is logged into your app, you know which practitioner or clinic staff member is performing the operation. Remember that session grant code you implemented earlier in your back end? Go ahead and use that method to get a session grant and related secret token for your user.

FYI

When the secret token belongs to a clinic staff member (someone who's Fullscript user type is 'Staff') or to the user configured as the store owner, the feature shows a practitioner picker. This way, they can select which practitioner the treatment plan is written on behalf of.
It's OK if you don't know which Fullscript patient the treatment plan is for. You can pass along information you know about the patient, and Fullscript searches for a match. The feature will show your user a list of patient matches, and give them the option to create a Fullscript patient record one if no match is found or if the practitioner decides none is the correct match.

Tip

When it comes to specifying the patient, you can rely on searching as proposed above, then saving the resulting id for the next time, or you can use Fullscript RESTful APIs such as `clinic/search/patients` to search behind the scenes to get the Fullscript patient id.

Initialization parameters

`secretToken` **REQUIRED**

A single-use token that is used to authenticate the practitioner or clinic staff member using the platform feature. Retrieved from the Fullscript RESTful API, see Codify user session grants.

`patient` **OBJECT**

An object describing the patient. Specify the patient’s id for an exact match, or some of the other parameters to trigger a search. Leave empty to prompt the user to create a new patient.

Child attributes:

- `id` **(string)** Fullscript patient id. If specifying the id, there’s no need to specify other attributes - they are ignored, even if they differ from what’s stored with Fullscript.
- `firstName` **(string)** The patient’s first name.
- `lastName` **(string)** The patient’s last name / family name.
- `email` **(string)** The patient’s email address.
- `dateOfBirth` **(string)** The patient’s birth date, in the format yyyy-mm-dd.
- `biologicalSex` **(string)** The patient’s assigned sex. Options are: male, female, or prefer not to say.
- `mobileNumber` **(string)** Patient's mobile number in the format +12223334444.
- `discount` **(number)** Patient discount level (in percentage). This discount does not include the clinic discount. Defaults to 0.

`entrypoint` **Optional**

This option lets you specify which page the user will land on after the practitioner and patient selector. The available options are `catalog` and `labs`*, with `catalog` set as the default if no entrypoint is specified.

- `catalog` **(default)**: Starts the user on the product catalog page.
- `labs`*: Starts the user on the labs page.

*Available in the US only.

Example treatment plan initialization

Let’s look at a couple examples for initializing the treatment plan, and how the feature behaves based on the detail level given for the patient.

Treatment plan with a Fullscript patient id

```
const platformFeature = fullscriptClient.create("platform", {
  secretToken: "xxxxx",
  patient: {
    id: "xxxxx",
  },
});
```

If there’s a match on patient id, the patient selection screen is skipped and the treatment plan explicitly sets the patient to the one specified. Unlike other ways to match the patient, this method doesn’t trigger the `patient.selected` event. If there’s no match on patient id, the patient selector is shown and the user can search for or create a new patient.

FYI

If there’s a match on a specified patient id, all other patient data is ignored. Even if your data differs from the data Fullscript has for this patient. The practitioner can update the patient’s details in the platform feature.

Treatment plan with some patient data

```
const platformFeature = fullscriptClient.create("platform", {
  secretToken: "xxxxx",
  patient: {
    firstName: “Amos”,
    lastName: “Burton”
  },
});
```

Fullscript tries to find a patient match based on the provided data. If one (or more) is found, suggestions are shown to the user. They can choose a suggested patient or create a new one. If the user creates a new Fullscript patient, the platform feature auto-populates the new patient form with the data you used in your search.

Tip

For a seamless integration, be sure to **send all applicable patient information**. This avoids unnecessary manual entry and also stops the practitioner from looking it up elsewhere or asking the client for information they should already have.

When the patient is selected or created, you’ll receive a `patient.selected` event with their Fullscript patient id.

Treatment plan with no patient data

```
const platformFeature = fullscriptClient.create("platform", {
  secretToken: "xxxxx",
});
```

The user is prompted to create a new patient. You’ll receive a `patient.selected` event with the new Fullscript patient id.

Specifying entrypoint

You can specify which page the user will land on after selecting a practitioner and patient by setting the `entrypoint` option.

```
const platformFeature = fullscriptClient.create("platform", {
  secretToken: "xxxxx",
  entrypoint: "catalog", // "labs" is also available
});
```

If `catalog` is specified or no entrypoint is set, the user will start on the product catalog page after the practitioner and patient selector.

If `labs` is specified, the user will start on the labs page instead.

Mount and unmount

Now that everything is set up you can mount the feature with `mount()`. This must be done only after your mount point (created above) is loaded. So we suggest mounting it when your app receives the `DOMContentLoaded` event.

```
document.addEventListener("DOMContentLoaded", function () {
  platformFeature.mount("fullscript-embed-container");
});
```

When you’re done with the feature, you can remove it with `unmount()`, or you can navigate away from the page.

```
platformFeature.unmount();
```

Sizing features

Fullscript Embed features fill the parent container’s available space. This means that if the mount point you create has a width and height of 900px, the embedded feature will have a height and width of 900px.

FYI

Fullscript Embed features are fully responsive and will reflow if their size changes. For the best performance, we recommend a minimum viewport size of 768 × 1024.
Here’s a fixed size example:

```
<div style="width: 900px; height: 900px;" id="fullscript-embed-
container"></div>
```

If you want to embed the feature into a modal and your modal has strict sizing, you can set the mount point `div` to take all available space in the modal with the following:

```
<div id="my-modal">
  <div style="width: 100%; height: 100%;" id="fullscript-embed-
container"></div>
</div>
```

If you want Fullscript Embed to take the full width and height of your page, you can use the following viewport-based units:

```
<div style="width: 100vw; height: 100vh;" id="fullscript-embed-
container"></div>
```

Handle events

Your integration could be considered complete now that you have the feature loaded on the page. However, there’s more you can do to create a seamless user experience.

Fullscript Embed fires events when actions happen that your app may want to know about. For example, an event is fired when a practitioner finishes creating a treatment plan for a patient. The event includes data about the treatment plan that your app can store. When receiving this event, you’ll also know the user is done with the feature and you can choose to unmount it.

FYI

Unmounting is optional. It removes the feature from your page, but you can also leave it there, or navigate away if that makes sense for your use case.

Take a moment to review the available events for your feature.

Begin listening for events

To listen for the events fired by the embedded feature, subscribe with `platformFeature.on()`.

Listen for patient.selected (event)

Create a function to call when the `patient.selected` event is triggered. Connect this to the event with `platformFeature.on()`.

```
function handlePatientSelected(data) {
  console.log("data", data);
  // Use this as an opportunity to save the Fullscript patient id,
  // unmount the feature, change page etc
}

platformFeature.on("patient.selected", handlePatientSelected);
```

Or, use TypeScript (if using NPM version) to add the EventListener:

```
const handlePatientSelected = (data:
EventListenerPayload<"patient.selected">) => {
  console.log(data);
  // Use this as an opportunity to save the Fullscript patient id,
  // unmount the feature, change page etc
};
```

Listen for treatmentPlan.activated (event)

Create a function to call when the `treatmentPlan.activated` event is triggered. Connect this to the event with `platformFeature.on()`.

```
function handleTreatmentPlanActivated(data) {
  console.log("data", data);
  // Use this as an opportunity to store data about the plan,
  // unmount the feature, change page etc
}

platformFeature.on("treatmentPlan.activated",
handleTreatmentPlanActivated);
```

Alternately, use TypeScript (if using NPM version):

```
const handleTreatmentPlanActivated = (data:
EventListenerPayload<"treatmentPlan.activated">) => {
  console.log("data", data);
  // Use this as an opportunity to store data about the plan,
  // unmount the feature, change page etc
};
```

Stop listening for events

If desired, you can also stop listening to an event:

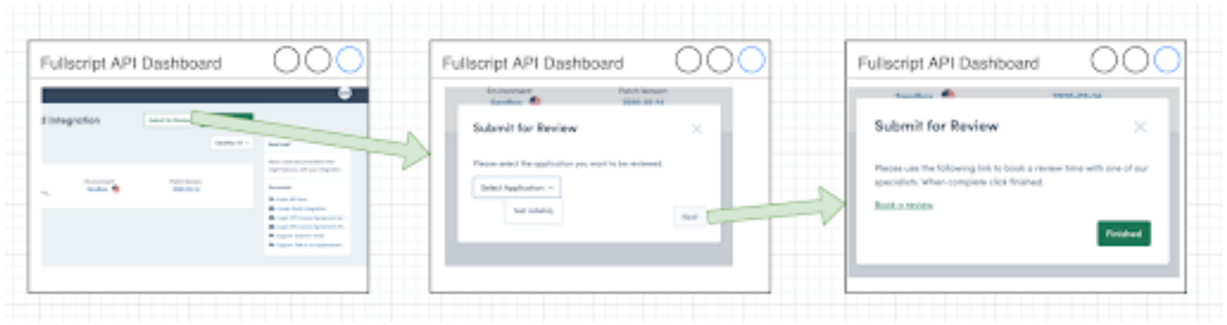
```
platformFeature.off("treatmentPlan.activated",
handleTreatmentPlanActivated);

platformFeature.off("patient.selected", handlePatientSelected);
```

Go live

Before creating production versions of your applications and going live, there is a review and approval process of your sandbox application. This requirement is also reflected in the Fullscript API License Agreement.

To get started, sign in to the Fullscript API Dashboard and click the **Submit for Review** button at the top of the page. In the two popup screens that appear, select the Sandbox application you want reviewed, then use the provided link to book a review time with one of our specialists.



Please select the application you want to be reviewed.

Please use the following link to book a review time with one of our specialists. When complete click finished.

We may have feedback and suggestions for you! Once this process is complete, you’ll be allowed to create and configure your production App(s) on the Fullscript API Dashboard. The final go-live process is influenced by contracting and agreed upon marketing efforts. You can request your production application to be listed in our in-app Integrations page via the dashboard also.

Go-Live Development Checklist

Complete development and testing of your sandbox application(s)
Submit your application for review and book a time with one of our specialists
If needed, address feedback or suggestions from the review
Create your production App(s) on the Fullscript API Dashboard

- be sure to provide your production OAuth redirect URI and configure the required OAuth scopes
- **Fullscript Embed:** provide your production Origin URI
- **Webhooks:** provide your app's production endpoint for webhooks

Update your app's production version code to point to production Fullscript API endpoints (US and/or Canada)
Update your app's source to use production OAuth client ID and secret values, found in your app's OAuth tab on the Fullscript API Dashboard. This value is different for each target environment (US vs Canada).
Fullscript Embed: Update your app's source to use production public key value, found in your app's Fullscript Embed tab on Fullscript API Dashboard. This value is different for each target environment (US vs Canada).
Webhooks: Update your app's secret challenge token value, and if verifying the validity of the incoming messages, also update the Fullscript signature secret key. These values are different for each target environment (US vs Canada).
Whitelist Fullscript domains (when applicable): If you have an existing Content Security Policy (CSP) implemented for IFrames, URLs, or scripts (if using UMD method) within your app, you have to whitelist the following domains to load `Fullscript Embed` iframe properly:

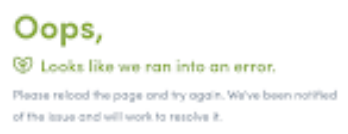
- `https://us.fullscript.com`
- `https://ca.fullscript.com`

Go-live 🎉: Request to have your integration listing published on Fullscript's in-app integrations page and arrange co-marketing efforts with the Fullscript team.

Error handling

If Fullscript Embed is not configured properly, or is attempting to access resources it's not permitted to, an error page is rendered and the error details are sent to the browser's console.

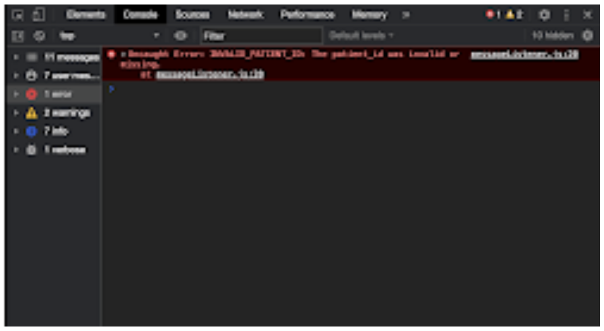
Errors look similar to this: (**left**: browser view. **right**: console window.)



Oops,

Looks like we ran into an error.

Please reload the page and try again. We've been notified of the issue and will work to resolve it.



```
Uncaught Error: INVALID_PATIENT_ID: The patient_id was invalid or missing.
    at messageListener.js:28
```

Here are the most common errors you may encounter with instructions for resolving them.

- `INVALID_SECRET_TOKEN: The secret_token was invalid or missing.`
  The session grant used to authorize the user is missing or invalid.
  - Verify that you're using the secret token from a fresh session grant (they are one-time use).
  - Verify that the secret token isn't expired (remember, it has a 120 second timeout).
  - Confirm that you're retrieving the session grant from the same Fullscript target environment that you're trying to use it with.
- `Refused to frame 'http://localhost:3000/' because an ancestor violates the following Content Security Policy directive: "frame-ancestors 'none'".`
  There are a few reasons this can happen.
  - Verify that your publicKey is valid and is for the correct Fullscript target environment
  - Check that your domain and port number exactly match one of the domains in the "Origin URI" whitelist you set in the Fullscript API Dashboard.
- `Error: Could not find the mount point for the iframe.`
  Fullscript Embed failed to find the mount point you added to your DOM.
  - Is the mount point's `id` spelled correctly? And is there just one instance of it?
  - Have you waited for the DOM to load before trying to find the mount point?
- `A CORS (cross-origin resource sharing) error when accessing the OAuth authorize endpoint.`
  Ensure you're not trying to `fetch()` the Fullscript OAuth url. Instead, redirect the browser to the `authorize` endpoint.

Events

Here are the events each feature triggers.

Platform

Event	Description
`patient.selected`	Fired when a practitioner creates or selects a patient. The data returned identifies the selected patient. You can store the Fullscript patient id with your local patient data to avoid triggering a search next time a practitioner creates a treatment plan for this patient. Event details found below.
`treatmentPlan.activated`	Fired when the practitioner or staff member finalizes the treatment plan. The data returned includes details about who created the plan, for which patient, and an array with the selected products and their details. Event details found below.

Event details for patient.selected

Here is a TypeScript definition for the `patient.selected` event data. For more information about possible values and formats, refer to the Fullscript Patient object in our REST API docs.

```
declare type PatientPayload = {
  patient: {
    id: string;
    firstName: string;
    lastName: string;
    email: string;
    dateOfBirth: string;
    biologicalSex: string;
    discount: number;
    totalDiscount: number;
    mobileNumber: string;
    textMessageNotification: boolean;
  };
};
```

Event details for treatmentPlan.activated

Here is a TypeScript definition for the `treatmentPlan.activated` event data. For more information about possible values and formats, refer to the Fullscript TreatmentPlan object in our

REST API docs.

```
declare type TreatmentPlanPayload = {
  treatmentPlan: {
    id: string;
    state: string;
    patient: {
      id: string;
      firstName: string;
      lastName: string;
      email: string;
    };
    practitioner: {
      id: string;
    };
    availableAt: string;
    recommendations: {
      variantId: string;
      refill: boolean;
      unitsToPurchase: number;
      dosage: {
        recommendedAmount: string;
        recommendedFrequency: string;
        recommendedDuration: string;
        format: string;
        additionalInfo: string;
      };
    }[];
    lab_recommendations: {
      id: string;
      name: string;
      requires_fasting: boolean;
      instructions: string;
      tests: {
        id: string;
        name: string;
      }[];
    }[];
  };
};
```

Session grant endpoint

Session grant object

`id` **string**

Uniquely identifies this session grant.

`claimed_at` **datetime**

The moment this session grant was used to sign in as a Fullscript user.

`expires_at` **datetime**

Time the session grant expires at. Set to 120 seconds from the creation. Must be used in that time frame.

`secret_token` **string**

The string you'll use to authenticate the Fullscript user when initializing your embeddable Fullscript Embed feature.

`user` **hash**

Child attributes

- `id` **(string)** The user's unique Fullscript identifier (uid).
- `type` **(string)** Type of user. One of `Practitioner` or `Staff`.

HTTP Request

```
POST https://api-us-snd.fullscript.io/api/clinic/embeddable/
session_grants
```

Header

`Authorization` **Required**

`Bearer XXXXXXXXXXXXXXXXXXXX`
Where XXXXXXXXXXXXXXXXXXXX is the current user's OAuth access token.

`Content-Type`

`application/json`

Arguments

`user_id`

The user’s unique Fullscript user identifier (id). Required only when using clinic-level OAuth. Newer applications that use role-based OAuth don’t need to provide this, because it’s contained in the access token.

`user_type`

`Practitioner` or `Staff`. Required only when using clinic-level OAuth. Newer applications that use role-based OAuth don’t need to provide this, because it’s contained in the access token.

HTTP response

Returns a Session Grant object.

Example request

```
curl -X "POST" "https://api-us-snd.fullscript.io/api/clinic/
embeddable/session_grants" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Example response

```
{
  "id": "aa43f1b7-8c3c-4dcb-9d49-e5c28601b10e",
  "secret_token": "yLGAfKXXXXz7LZufJRIIXXXXVCX0bXXXXpBVLXXXXZiEchU",
  "claimed_at": null,
  "expires_at": "2019-01-24T18:06:44.000Z",
  "user": {
    "type": "Practitioner",
    "id": "x1x0196x-5615-4874-xxx4-48x459180x09"
  }
}
```

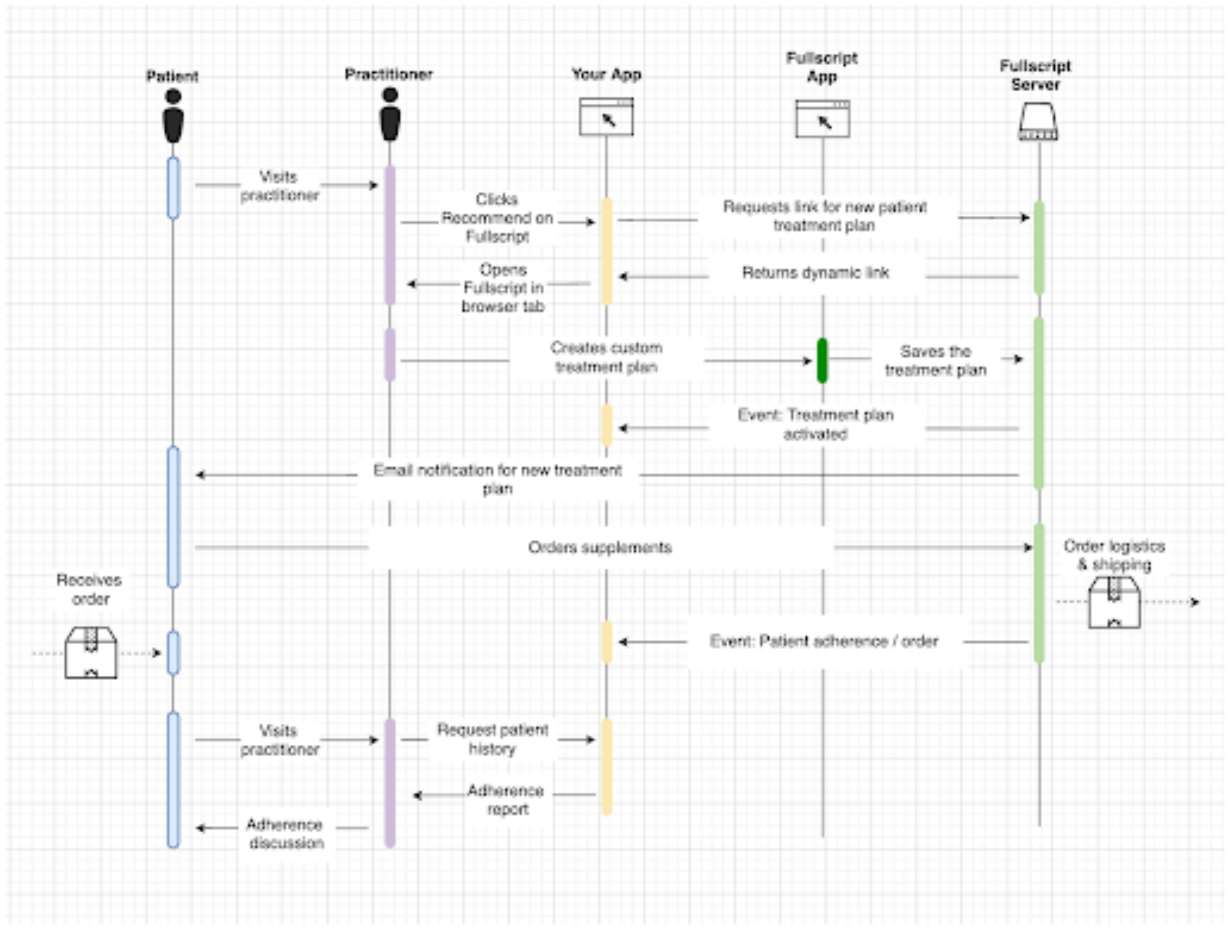
What is Fullscript Redirect?

For example, your app can include a “Create a Treatment Plan” button that opens a new browser tab to the patient’s record within the practitioner’s Fullscript account. Practitioners can search for and recommend supplements from the extensive Fullscript catalog, without venturing far from your app. When the plan is activated, or when the patient fills their order, our events system alerts your app so you can update their file for the practitioner.

Fullscript Redirect gives practitioners access to the Fullscript experience via a browser tab, opened from a button in your app.

You add a **“Recommend with Fullscript”** button to your app’s practitioner screens such as patient summary or patient interaction page. When clicked, your app opens a new browser tab with the same patient in Fullscript. The practitioner creates a Fullscript treatment plan then heads back to your app.

In addition, your app can give an end to end picture of patient care by subscribing to relevant events that happen in Fullscript. For example, when your app listens for the treatment plan “created” event, you’re notified and will receive the full plan details which you can display or store in your app.


Patient
Practitioner
Your App
Fullscript App
Fullscript Server
Visits practitioner
Clicks Recommend on Fullscript
Requests link for new patient treatment plan
Opens Fullscript in browser tab
Returns dynamic link
Creates custom treatment plan
Saves the treatment plan
Event: Treatment plan activated
Email notification for new treatment plan
Orders supplements
Order logistics & shipping
Receives order
Event: Patient adherence / order
Visits practitioner
Request patient history
Adherence report
Adherence discussion

Integration process overview

Here are the big picture steps you'll follow with a Fullscript Redirect integration. More info for each of these steps is found later in this guide.

1 Design & Decisions

2 Setup

3 User authentication

4 Add the treatment plan dynamic link

5 Optionally: Improve the practitioner and patient experience

6 Go Live

Important links

Here are the links you're most likely to use as you develop your Fullscript integration.

Fullscript API dashboard

Your primary destination to register App instances with our API, get OAuth setup details, register for webhooks, and more.

- Go to the Fullscript API Dashboard

API reference docs

Find a detailed listing of all our API endpoints. Includes examples, inputs, outputs, and data object definitions.

- Technical reference

Development sandboxes

API endpoints to use during development. Tests your integration without affecting live patient records. You can use the Create Application function in the Fullscript API Dashboard to create multiple test Apps per target (US and Canada). This can be helpful for testing and debugging different builds of your app.

- US: `https://api-us-snd.fullscript.io/api 🇺🇸`
- CA: `https://api-ca-snd.fullscript.io/api 🇨🇦`

Production servers

Live production API endpoints. Only for use with production software. You'll be able to create production applications once your sandbox integrations have been approved by our team.

- US: `https://api-us.fullscript.io/api 🇺🇸 🌐`
- CA: `https://api-ca.fullscript.io/api 🇨🇦 🌐`

Development sandbox

Development happens against our sandbox servers. Before you go live, we have a review process that gives you access to our production servers.

Our sandbox environments are nearly identical to production. As you read these docs, you'll notice we highlight any differences you need to be aware of.

To get you started, here are the top three sandbox vs production considerations:

1. Sandbox and production servers have different target URLs to use in your API calls. See Important links for details.

   FYI

   Regulations and our product catalogs vary slightly between the USA and Canada, so we have dedicated servers (sandbox and production) for each. If your software targets practitioners in both the USA and Canada, you need two copies of your App. One for each target.

2. Sandbox Apps have OAuth keys and secret strings that are unique and will be different in your production App.
3. For obvious reasons, our sandbox environment doesn't process actual patient supplement orders. However, you can place test orders using Stripe's test credit card:

   ```
   4242 4242 4242 4242 (with any future expiry date)
   ```

4
5

Tip

To simplify your switch from development to production, we suggest storing server-specific info, such as the base target URL and your OAuth secret keys, in environment variables or using another easy to swap technique.

At the end of this guide, you'll find a Go Live checklist to make sure your launch day goes smoothly.

Design decisions

There are some common design questions that will come up as you work through your Fullscript integration. We've tackled these below to give you a starting point for your discussions.

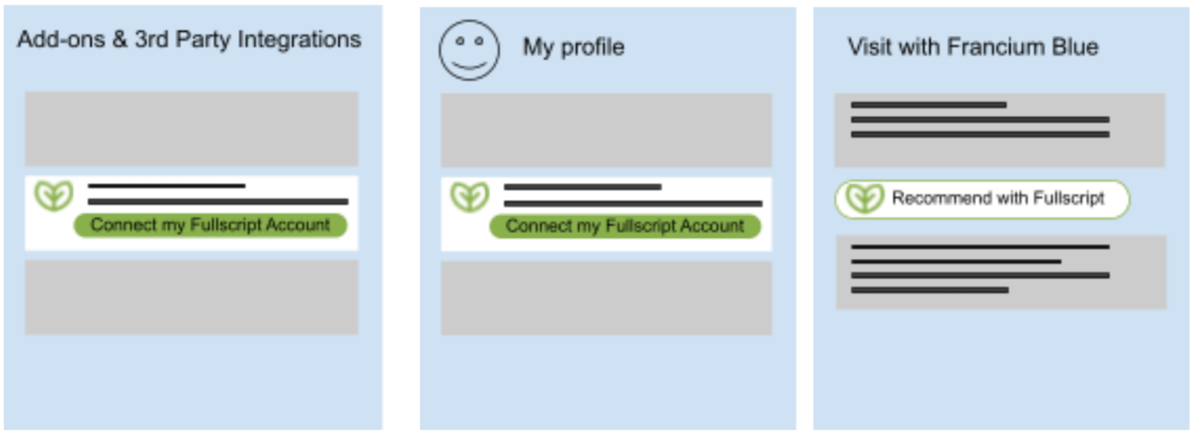
How will users discover the Fullscript integration?

Fullscript APIs use role-based OAuth, which means each user authorizes their own Fullscript account access - even if all the accounts are part of the same Fullscript clinic.

Most of our API calls require the user's OAuth Bearer token to identify who you're making the call on behalf of. Getting that authorization is one of the first design issues to address.

The three most common ways our partners make the integration available to their users are:

- on a third party plugins page
- on the user's profile page
- by triggering the OAuth flow the first time the user interacts with the integration


Add-ons & 3rd Party Integrations
Connect my Fullscript Account
My profile
Connect my Fullscript Account
Visit with Francium Blue
Recommend with Fullscript

If your app already has a section for integrations or third party plug-ins, your users know where to go looking for add-ons. Add a **"Connect my Fullscript account"** button that triggers the user's OAuth flow. On patient interaction pages, show or hide the **"Recommend with Fullscript"** button depending whether you have an auth token for the current user.

Alternatively (or even in parallel) you can choose to always show the **"Recommend with Fullscript"** button on the page(s) where it's relevant. If the user hasn't already completed the OAuth process, simply route them there before continuing with the normal button operation. This is a great solution for systems that don't have a dedicated integrations page, or who want to make the Fullscript integration very visible.

FYI

There is no API available to “toggle” the integration’s availability for all users. If your design requires that a particular user enables (or disables) the ability to use the Fullscript integration, you’ll need to create your own toggle. Each user will still need to proceed with their own OAuth authorization.

How will users revoke or remove the Fullscript integration?

As part of your integration, make sure you include a place for users to revoke their OAuth authorization. Not only is this the inverse to letting users authorize access, it’s also super helpful for testing your integration. As a developer, you will no doubt use this feature more than any real user!

Generally, our partners put this feature on their third party integrations page or in the user’s profile.

We also suggest you add:

- a master “revoke all Fullscript user authorization” button that’s accessible to developers and admin users
- a way for admins to remove a selected user’s authorization for employee lifecycle management

Tip

No bearer token is needed to revoke a token. You only need your app’s client ID, client secret, and the token to revoke.

FYI

Practitioners can also revoke authorization from the Fullscript Web App.

How much to cross-index between systems?

To search for a patient or a practitioner via the Fullscript API, you often provide the user’s email address. This gives you the fastest match because email address is a unique value in our system.

However, particularly in the case of patients, we find that some of our users share an email address. For example, a patient’s guardian may use their email address for orders for their charge and for themselves - meaning that two or more people share the same Fullscript patient account.

This can make it harder for your app to match a patient in your system with one in ours. To simplify, certain resources in our system have an available `metadata` field where you can store your system’s unique identifier. Then you can use the metadata endpoint to search our system for those resources later.

The following objects accept metadata when being created or updated:

- `patient`
- `practitioner`
- `staff`
- `treatment_plan`

Additionally, some of our patient-related REST API calls require the Fullscript patient id (which you can find with search for patients. If this becomes too taxing, you may choose to store our patient id with your records so you can skip the search step the next time.

When designing your system, you can choose none, one, or both of these cross-indexing options.

Tip
Remember that in the Fullscript account model, patients are owned by a clinic. If your app supports multiple clinics, a single patient record in your application could be tied to multiple Fullscript clinics. Each of their Fullscript accounts has a unique email address.

Will you store or fetch Fullscript data (or both!)?

Depending on how much Fullscript data you show users, you may wish to store Fullscript data locally rather than use the API to fetch it every time. There may also be key user interaction pages where you start with what you have, but always want to fetch the latest data.

For example, let’s consider a patient history page where you plan to display previous encounters and any resulting Fullscript treatment plans. You could fetch this data every time you need it, waiting to fill those sections of the page as the data is retrieved. Or, you could show the data you have on hand, then verify with Fullscript if anything new or changed since you last fetched it, and update if necessary.

Similarly, if you’re integrating with Fullscript Redirect, you can update the patient page with the latest data after the practitioners saves the new treatment plan. For more details, see our Fullscript Redirect guide to updating the patient encounter page.

There are three ways to get Fullscript data:

- fetch the current data via the applicable API endpoints - this provides your app information as-needed
- register for webhooks and receive updates to data of interest to you - some partners use

this for on-demand updates only, others use it to store event data for later

- set up a regular job to query our events endpoint - used to store data for later

FYI

Webhooks are triggered for actions your practitioner takes through the Fullscript Redirect URL, but **not** for anything you do specifically through the API. For example, if you use the `patients` endpoint to update the spelling of a Fullscript patient's name, we won't alert you of this since we're pretty sure you already know!

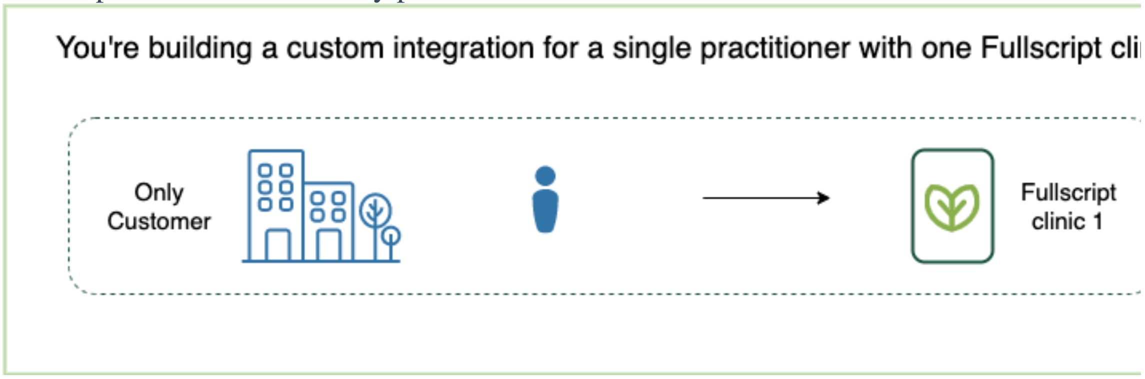
How will you support multiple practitioners?

When designing your interactions with Fullscript, it's helpful to understand our clinic and user account structure.

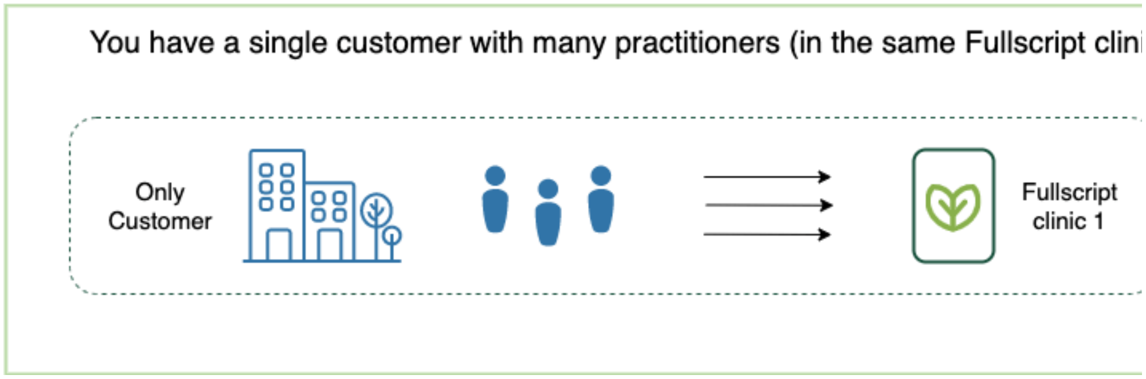
Fullscript divides its users into what we call **clinics**. A clinic has an **owner**, one or more **practitioners** (including the owner), **staff members** (also called **clerks**) who enter data on behalf of practitioners, and a set of **patients** with related **treatment plans**. Each practitioner and staff member has their own Fullscript account, with login tied to their email address.

We find there are four common use cases when it comes to integrating with Fullscript.

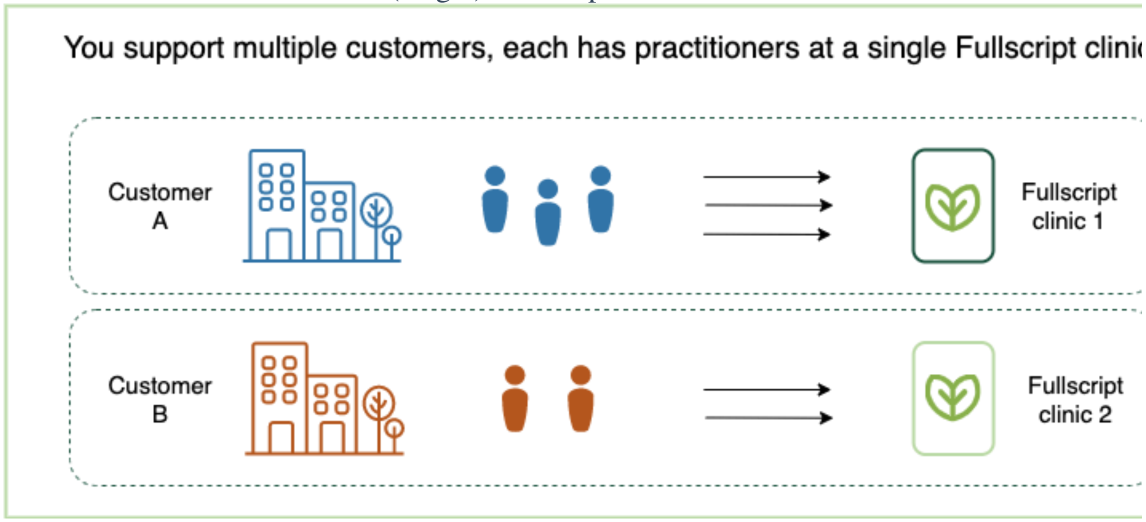
1 You're building an integration for one customer. Your customer is the owner of a single Fullscript clinic and is the only practitioner.


You're building a custom integration for a single practitioner with one Fullscript cli
Only Customer
Fullscript clinic 1

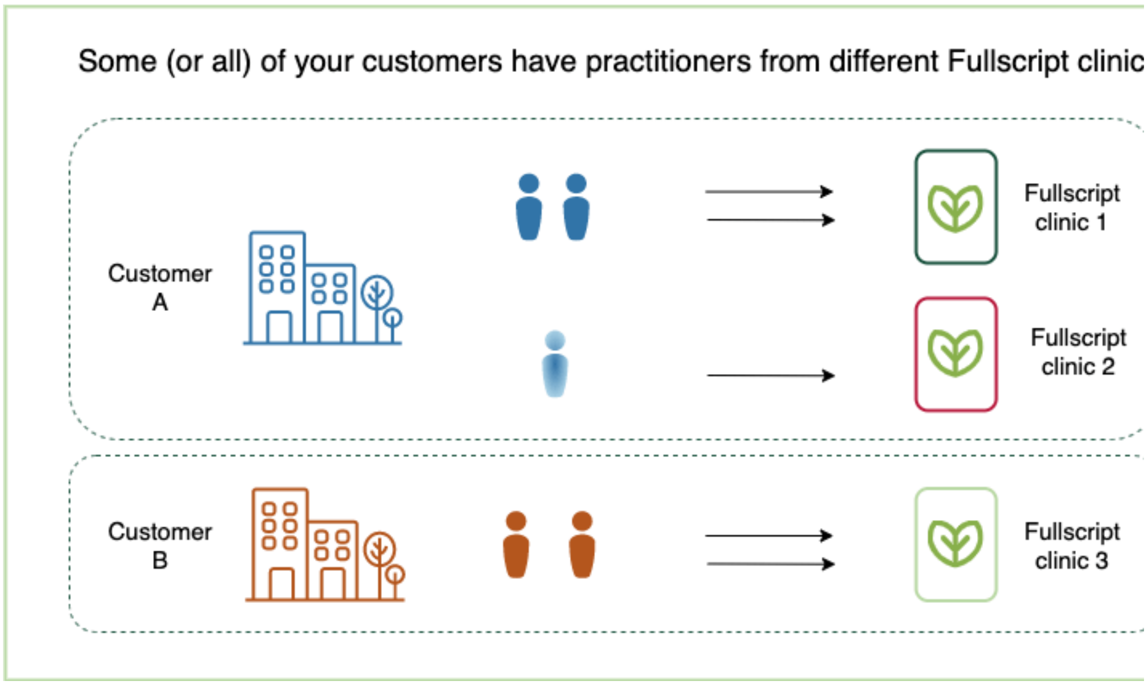
2 You're building an integration for one customer who owns a single Fullscript clinic. This clinic has multiple practitioners.


You have a single customer with many practitioners (in the same Fullscript clini
Only Customer
Fullscript clinic 1

3 You’re building a system to be used by multiple customers. Your customers’ Fullscript accounts are each in their own (single) Fullscript clinic.


You support multiple customers, each has practitioners at a single Fullscript clini
Customer A
Fullscript clinic 1
Customer B
Fullscript clinic 2

4 Your system can be used by one or more customers. Each customer may have one or more Fullscript clinics.


Some (or all) of your customers have practitioners from different Fullscript clinic
Customer A
Fullscript clinic 1
Fullscript clinic 2
Customer B
Fullscript clinic 3

It's useful to note that Fullscript clinics and accounts are free to create. Each account simply needs a unique email address. Given it's free to make multiple accounts, some practitioners have accounts at different Fullscript clinics.

For example, a practitioner may work for two different organizations and have an account with the Fullscript clinic associated with each of those companies. Or, a practitioner may have one Fullscript clinic configured for use with a philanthropic practice, and another for their main practice.

Tip

If your app supports multiple Fullscript users (regardless of whether that's through a single clinic or multiple clinics), there's no requirement that the Fullscript accounts are pre-created. Your customer's Fullscript account owner can pre-invite their users through the Fullscript Web App, but users can also simply sign up during your app's OAuth process.

Supporting a single Fullscript clinic

This is the simplest integration path. Regardless of which of our first two use cases you're building for, you're only supporting a single Fullscript clinic. The clinic has one or more Fullscript user accounts (practitioners and staff members).

- With only one clinic to manage, there's no concern for which clinic a practitioner, patient, or event belongs to.

Supporting multiple Fullscript clinics

This is the case when your app has multiple unrelated sets of customers. It matches with our last two use cases above.

- Your app may need to keep track of which clinic a practitioner or patient belongs to. For example, you may have a reason to keep a list of practitioners per Fullscript clinic. To know which clinic a practitioner or staff member belongs to - use the retrieve a clinic endpoint with their OAuth token. Store the `clinic_id` with the user's data so you know which clinic is being accessed for future calls made with the user's OAuth token.
- If your app signs up for Fullscript webhooks or uses our events endpoint, your app may need to know which clinic the activity occurred with to know which set of data to update. Find this in the payload's `clinic_id` parameter.

Practitioner workflow planning

The basic practitioner workflow

Typically, integrations with Fullscript Redirect use a flow similar to this:

1. The practitioner logs into your app and navigates to a patient screen with the Fullscript treatment plan integration.
2. They click the **Recommend on Fullscript** button; a new browser tab opens.
3. If needed, the practitioner signs in to their Fullscript account.
4. The practitioner sees their Fullscript dispensary, opened to the same patient they were just viewing in your app.
5. The practitioner creates, then activates a patient recommendation.
6. The practitioner closes the Fullscript browser tab and goes back to your app.

FYI

This flow assumes the practitioner has authorized your application's use of their Fullscript account and data. For more information on authorization, see the design discussion about authorization and our section on OAuth.

A very basic Fullscript Redirect integration stops at the list above. But there are many other ways you can integrate Fullscript with your app. Our REST API has a wealth of data available to improve the experience of practitioners and patients alike.

Extended practitioner workflow options

Here’s a list of great add-on experiences using our REST API. These suggestions keep practitioners working in your app and build a more comprehensive user experience.

- Update your patient record screens with details of treatment plans created in Fullscript.
- When displaying a patient’s record, list or display details of all their draft and active (or past) Fullscript treatment plans.
- Include a button to let the practitioner edit an active or draft treatment plan.
- Include a button that allows the practitioner to cancel a treatment plan.
- Show the practitioner a timeline of supplement purchases made by the patient to facilitate conversations about protocol adherence.
- Update patient information (records, supplement changes, etc) with updates that are made on Fullscript.

Read more about these extended integration options in Other useful info from Fullscript.

Patient workflow options

If your application includes a patient portal, take a look at these suggestions for improving the usefulness of your platform to patients using it:

- Share the patient’s mobile number with Fullscript, allowing Fullscript to send automatic mobile messages (SMS) with treatment plan notices and refill reminders.
- Add treatment plan list and/or details to the patient’s portal.
- Add a unique “purchase now” button to the patient’s dashboard, visit summary, or treatment plans page. You can even use it in email notifications.

We’ve measured the success of these methods, and it shouldn’t come as a surprise that adding treatment plan details to the patient portal is proven to increase plan adherence.

You’ll find implementation details for these options in Improve the patient experience.

FYI

There’s no checkout experience available via the API. The **purchase now** button redirects the patient to their Fullscript account for checkout. Our patient portal logic determines taxes and surface shipping options based on the patient’s location.

Setup

If you don't already have an **App** created with Fullscript, sign in to the Fullscript API Dashboard and click **Create Application** (or continue to sign up if you're brand new to us!). For initial development, target one of our sandbox servers (either US or Canadian).

You'll need an application name (currently used for your information only), and a redirect uri where we will send users once they have authorized your app's request to access their Fullscript data.

Tip

Both the application name and redirect uri can be edited later, so there's no need to agonize over those decisions now.

Configure scopes

The Fullscript Redirect integration process needs the following OAuth scopes:

- `Clinic:read`
- `Clinic:write`
- `Patients:write`
- `Patients:treatment_plan_history`

Request these scopes via the Fullscript API Dashboard's **OAuth scopes** settings for your **App**. Don't forget to save your changes.

Create test users

Create one or more test practitioners by clicking the **Create Test Account** button on the right side of your **App**'s page in the Fullscript API Dashboard. For now, choose **Skip** when asked to upload certifications for your new test users. Stop when you get to the authorization screen for your new user. It requires you to have your redirect endpoint available. We'll get to that in a few more steps.

FYI

When you create test practitioners as described above, they are automatically approved and there's no need to upload fake certifications for your test users.

You'll also need at least one test patient for your integration. Use the **Patient Sign Up** link from your practitioner's welcome email, or log in to the **Fullscript App** as the test practitioner, then select **Patients > New Patient**.

Here are the practitioner login links for the Fullscript sandbox Apps:

- US Sandbox: https://us-snd.fullscript.io/login
- Canadian Sandbox: https://ca-snd.fullscript.io/login

OAuth for user authentication

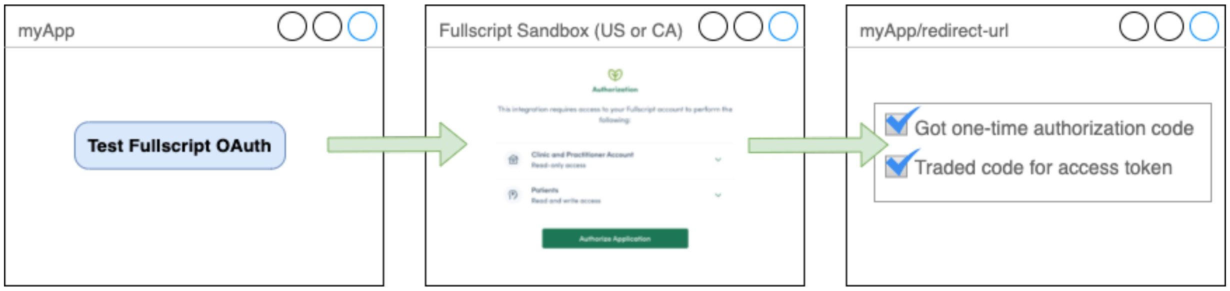
By default, apps don't have access to most functionality in the Fullscript API. Users own the data they've entered into Fullscript, and it's up to them to grant your app access to that data.

Fullscript uses the OAuth 2.0 "authorization code flow" for API access. Follow our OAuth tutorial to add and configure OAuth.

When you're done the tutorial, make sure your app stores the following info:

- `user's access_token`
- `user's refresh_token`
- `resource_owner > id`
- `resource_owner > type`

Create a simple test flow something like this to make sure that your app has OAuth working and your redirect endpoint properly enabled:


myApp
Test Fullscript OAuth
Fullscript Sandbox (US or CA)
Authorization
Authorize Application
myApp/redirect-url
Got one-time authorization code
Traded code for access token

Tip

If your application supports more than one customer or more than one Fullscript clinic, store the user's `clinic_id` (found by retrieving the clinic info, using their OAuth token as the Bearer token) with the practitioner's user data. Begin grouping Fullscript data together by the `clinic_id`.

Now that you've got the OAuth flow working, add it to your app in the place chosen during the design phase.

To make things easy, we have a pre-created button pack you can use. Or, use our logo files to create your own. Here are three of our most commonly used colors:

- green (#88B04B) 🟩
- coal (#2E3A47) ■
- forest (#1B533F) ■

Important
Before continuing, run through the OAuth process with one of your test practitioners (created earlier) to make sure your app can retrieve the user's `access_token`.

Treatment plan dynamic link

The secret sauce behind your Fullscript Redirect **"Recommend with Fullscript"** button is the dynamic link that you connect to it. The Fullscript APIs offer two possible links: **draft** treatment plan and **new** treatment plan.

For the best user experience, call our create a **draft** treatment plan endpoint and include patient information in your request. For a more generic experience where the practitioner finds or creates the patient themselves, use retrieve a **new** treatment plan to get a generic link for the current practitioner.

Either way, in the header of your request, include the practitioner or staff member's OAuth `access_token` as the bearer token. This identifies who is using Fullscript, and will ask them to sign in if they have signed out.

Important
Don't hard code your dynamic link urls, instead use the endpoint to get the `redirect_url` each time. The value specified in `redirect_url` can change (and is different between sandbox and production environments!)

Draft treatment plan link

Your application can support either of the following treatment plan creation scenarios:

1. a Fullscript **practitioner** is drafting a patient treatment plan

2 a Fullscript **staff member** is writing the draft plan on behalf of a practitioner

To get the draft treatment plan link, you need:

- the OAuth `access_token` (bearer token) for the current user; and
- the `practitioner_id` for the practitioner whose name will be on the treatment plan

To determine your user's Fullscript account type (practitioner or staff member), check the **current** user's `access_token` > `resource_owner` > `type`.

```
userType = token.resource_owner.type;
```

Find the applicable `practitioner_id` for your call by examining the **relevant** user's `access_token` > `resource_owner` > `id`.

```
if (userType == “practitioner”) {
   practitionerId = token.resource_owner.id;
}
```

You can optionally provide information about the patient the practitioner is working with. If you choose to do so, the patient's email address is **required**.

```
curl -X "POST" "https://api-us-snd.fullscript.io/api/clinic/
dynamic_links/treatment_plans" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX' \
  -d $'{
  "patient": {
    "email": "johndoe@fakemail.com"
  },
  "treatment_plan": {
    "practitioner_id": "x1x0196x-5615-4874-xxx4-48x459180x09"
  }
}'
```

If your query includes patient info, but Fullscript can't find a matching patient in your clinic, a new patient record is automatically created for you, and a welcome email will be sent to the patient.

FYI

Be careful about the email addresses you use during development and testing! Those addresses will receive a welcome email.

You can’t include the patient’s first or last name without an email address, or you’ll get an error due to incomplete data:

```
POST
https://ca-snd.fullscript.io/api/clinic/dynamic_links/
treatment_plans
Bearer Sax65b94DFHn1va6Y3F-VveBgABX-t8C-3qWyuS1eNI
422
{
   error: `["Last name can't be blank", "Email can't be blank",
"Email is invalid"]`
}
```

Whether the call creates a new Fullscript patient or matches with an existing one, the response includes the dynamic link where you’ll forward the practitioner.

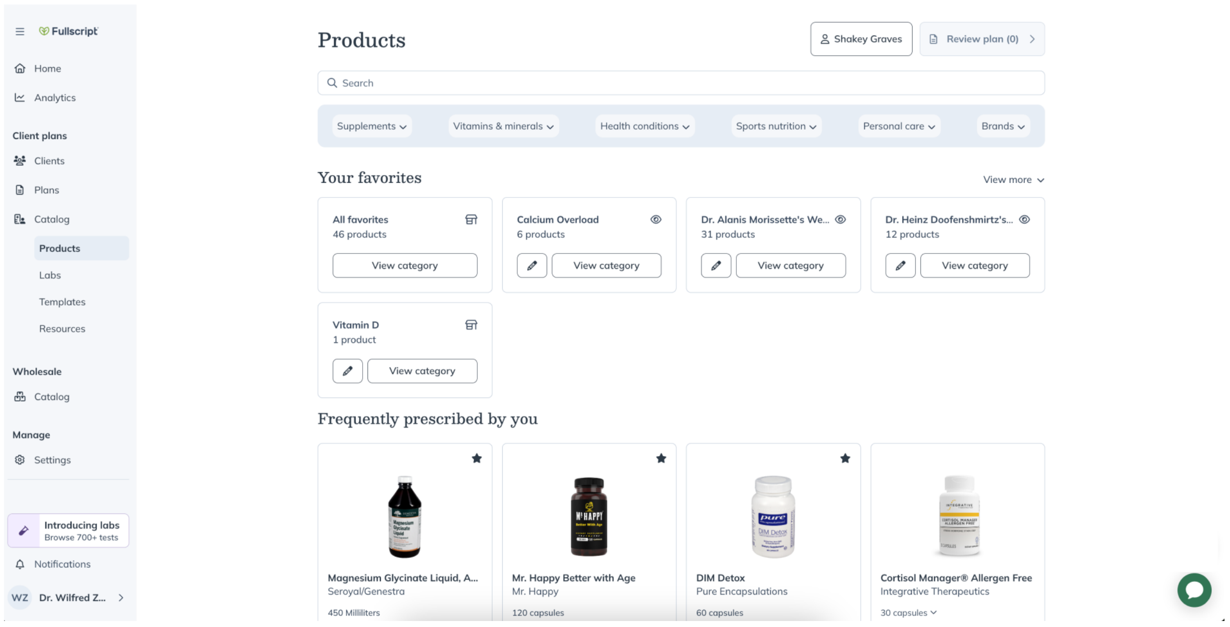
Find the `redirect_url` nested under `dynamic_links` in the draft treatment plan response.

```
redirectURL = response.dynamic_links.redirect_url;
```

Tip

There’s no time limit on this draft treatment plan link, so you can request it when the page loads or wait and only request it when the user clicks your **“Recommend with Fullscript”** button. Get a fresh new link after using this one though, because the link is tied to the treatment plan’s id and future use of the same link opens the same treatment plan, not to a new one.

If a patient record was created or found, the practitioner lands on the Fullscript patient record, ready to add a treatment plan. Should the practitioner feel the match is incorrect, they can use the dropdown toggle next to the patient name to switch the patient they are reviewing.



If no patient data was forwarded, the practitioner lands on a generic page (same as the new treatment plan link, below) where they're able to search or create a patient record on Fullscript.

Tip

Remember to set the dynamic link to open in a new tab so the practitioner can easily return to your app after saving the treatment plan.

New treatment plan link

The new treatment plan link is a `GET` method, so there's no patient data or other body information to provide.

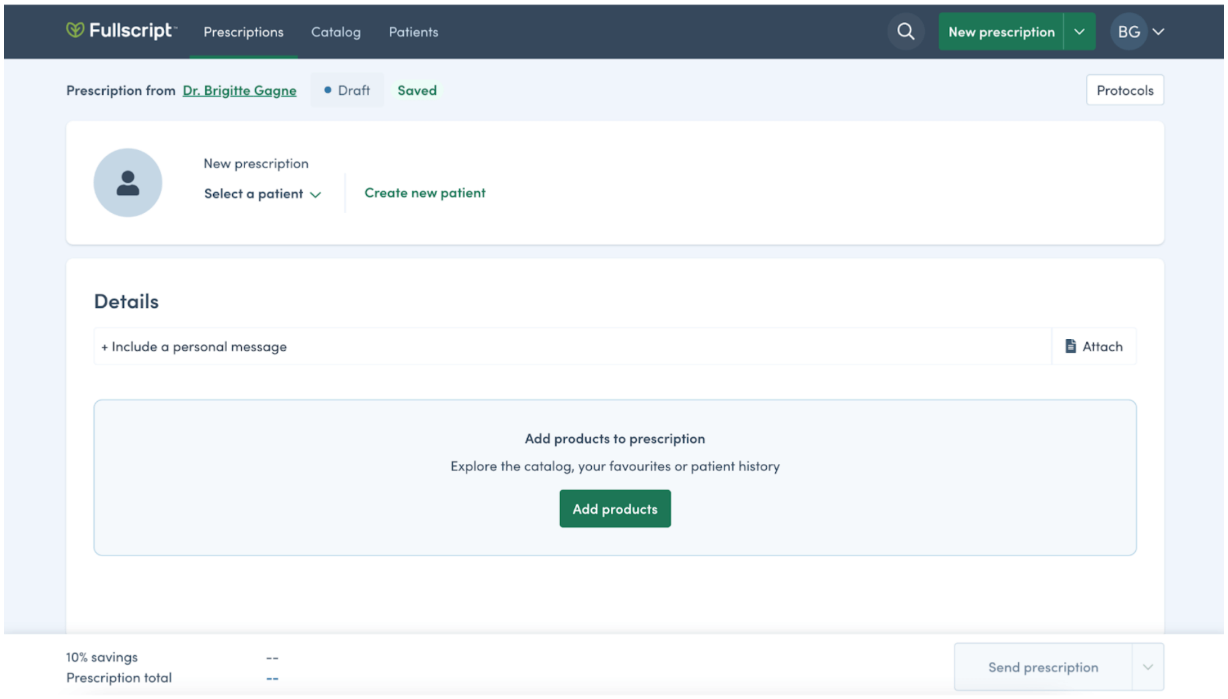
```
curl -X GET https://api-us-snd.fullscript.io/api/clinic/
dynamic_links/treatment_plans  \
   -H 'Content-Type: application/json' \
   -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX' \
}'
```

The response includes the dynamic link where you'll forward the practitioner.

Find the `redirect_url` directly at the top of the new treatment plan response.

```
redirectURL = response.redirect_url;
```

When the practitioner opens this link, they're presented with the option to create or find a patient.



Tip

Remember to set the dynamic link to open in a new tab so the practitioner can easily return to your app after saving the treatment plan.

Display the new treatment plan

Congratulations! With the steps above complete, you have your basic Fullscript Redirect integration working. Now, let's work on a simple augmentation to improve the practitioner experience by copying the treatment plan details to the practitioner's patient screen in your app.

To do this, we recommend subscribing to our webhooks which will automatically alert you of the new treatment plan. If webhooks won't work for your infrastructure, you can also use our REST API treatment plans endpoint.

Entrypoint

You can specify which page the `redirect_url` should target by setting the `entrypoint` option on certain dynamic link endpoints. If no `entrypoint` is specified, the user will start on the product catalog page by default. More details can be found in the techincal refrence.

```
curl -X "POST" "https://api.us-snd.fullscript.io/api/clinic/
dynamic_link" \
 -H "Content-Type: application/json" \
 -H "Authorization: Bearer XXXXXXXXXXXXXXXXXXXXXXXXXXX" \
 -d '{
  "patient": {
    "email": "johndoe@fakemail.com"
  },
   "entrypoint": "catalog"
}'
```

- `"catalog"` (default) → The user starts on the product catalog page.
- `"labs"`* → The user starts on the labs page. A treatment plan is automatically created for the user, and any selected products will be added accordingly.

*Available in the US only

Webhooks

Webhooks are a type of API where we push data to your app instead of you pulling it from Fullscript.

When your app is subscribed to the Treatment plan created webhook, you'll receive automatic notification of all new treatment plans. When you receive this notification, you can update the page the practitioner is viewing.

We have a handy webhook tutorial that provides everything you'll need to know about working with webhooks.

Treatment plan endpoint

If you can't or prefer not to implement webhook support, you can still show the practitioner their most recently created treatment plan using our REST APIs.

To do this, give the practitioner a button that requests a refresh of the patient data. While refreshing the data, call our treatment_plans API to find the latest treatment plan(s) for the patient.

Get the list of available treatment plans for a patient, and their respective `created_at` date, with `treatment_plans`.

Then retrieve the details of an individual plan with `retrieve-a-treament-plan`.

Tip

The call to request treatment plans requires the patient id. If your app uses the draft treatment plan link mechanism, the return for this call includes the patient id. Otherwise, obtain the patient id with a patient search.

Other useful info from Fullscript

Make your app stickier by giving patients and practitioners visibility into their Fullscript activities from your app. Our API provides a number of ways your app can listen for or request information your users want to see.

1. have the data pushed to you via our webhooks
2. request the data as you need it with our REST APIs
3. use our events endpoint to regularly fetch data event data from our system

Practitioner experience

By offering practitioners extra patient information in your app, you reduce the time they spend going to the **Fullscript Web App** to search for and find the same information. You're also making the patient's next visit a better one, because the practitioner has immediate access to the details from the past visit(s) and the patient's ordering activity since their last encounter.

Here are some of the practitioner experiences you can add.

Update the patient's encounter page after a treatment plan is created

Also described in Display the new treatment plan.

If you've registered for the Treatment plan created webhook, you'll be alerted nearly instantly of the practitioner activating a new treatment plan. When you receive this notification, update the patient interaction page.

If you don't want to use webhooks, you can have the user manually refresh the page; and use `patients/:patient_id/treatment_plans` to fetch the data for display.

Include a full list of the patient's treatment plans

It is helpful to show the practitioner the patient's currently available treatment plans. This practice invites a conversation with the patient to discover what's been working (or not working) for them.

Use the endpoint `patients/:patient_id/treatment_plans` to fetch a list of treatment plans. Follow up with calls to retrieve a treatment plan to get the details for display.

If you're storing Fullscript data in your app, you can use the events endpoint to regularly query for new treatment plan events and store these for display later rather than relying on an active connection.

Add a link to let the practitioner edit active treatment plans

Practitioners aren't limited to creating new treatment plans with Fullscript Redirect. They can also edit existing ones.

Use our `clinic/dynamic_links/treatment_plans/:id` endpoint to retrieve a redirect link directly to the treatment plan which can be opened in a new tab for editing.

Add a button to cancel a treatment plan

To cancel a treatment plan, provide its plan id to this endpoint `clinic/treatment_plans/:treatment_plan_id/cancel`.

Add a timeline of supplements ordered by the patient

You can improve patient interactions by giving practitioners data about when and how often their patient ordered supplements.

If your app is registered for webhooks subscribe for order.placed event notifications. You'll be alerted when a patient fills an order. Save this data for the next time the practitioner views the patient.

If you prefer to work on-demand, wait until a practitioner loads a patient's page, then use the endpoint `clinic/patients/:patient_id/orders` to find and display this patient's orders.

Alternatively, at a set interval (for example, nightly), use the `events/patients` endpoint to check for events that happened for the current patient and store them for future display.

Apply changes made outside your application

While most practitioners will access Fullscript through your integration, it's possible that some may still use the Fullscript Web App directly. In that case, you won't know to go search for data that's changed.

You can subscribe to webhooks to get notifications of those changes. Or, set an interval (for example, nightly), to use the `events` endpoint, checking for changes you want to record.

Patient experience

The Fullscript API is primarily built to accommodate a streamlined practitioner workflow. However, we also offer ways you can help patients get the most benefit from the treatment plans recommended by their practitioners.

1. If your system provides Fullscript with the patient’s mobile phone number, Fullscript sends the patient text messages with their recommendation and refill reminders.
2. Using our treatment plan endpoint, you can request details of the patient’s recommendations and an `invitation_url` that takes the patient directly to their Fullscript order page. Add an **“Order Supplements”** button to the patient’s dashboard or encounter summary to make ordering supplements a breeze.

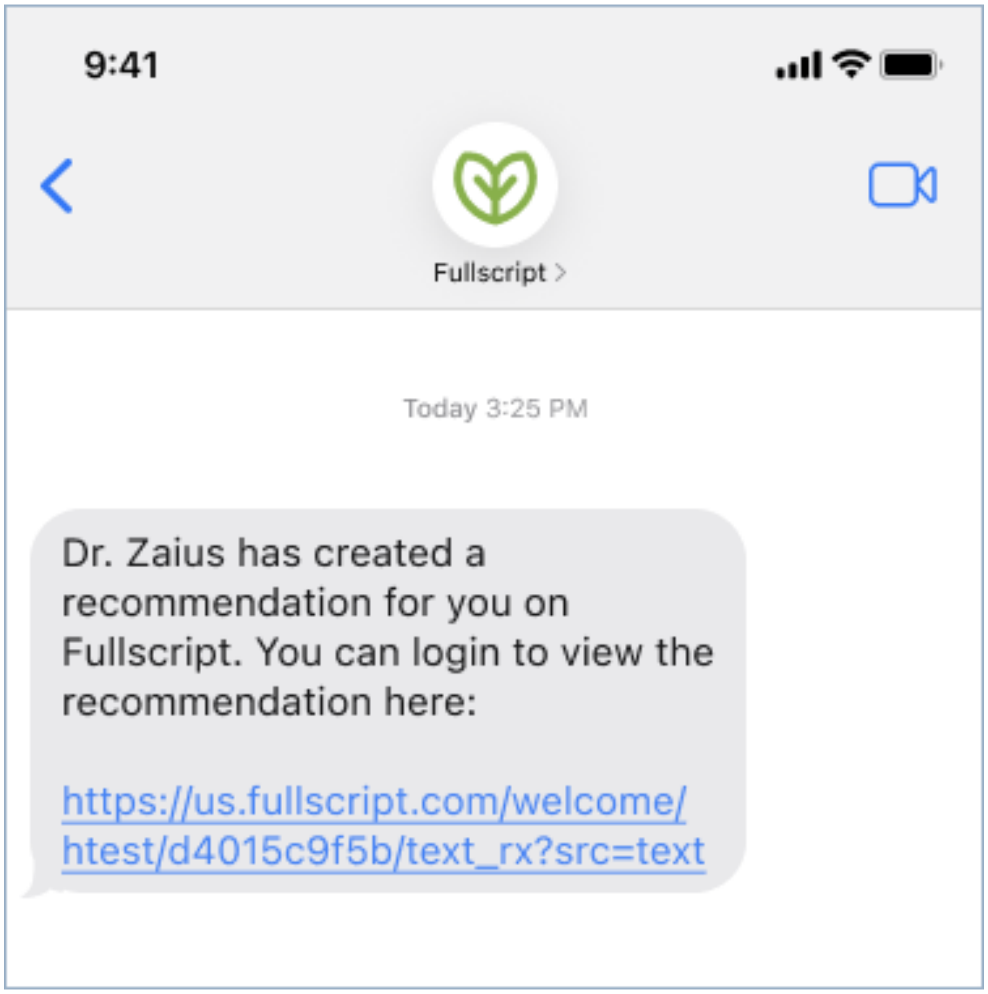
Both these methods are proven to increase patient adherence, which is a wellness target your practitioners want their patients to achieve.

FYI

There’s no checkout experience available via the API. The `invitation_url` redirects the patient to their Fullscript account for checkout. Our patient portal logic determines taxes and surface shipping options based on the patient’s location.

Patient mobile notifications

Text recommendations and reminders have been shown to significantly improve patient adherence to their treatment plans. This is particularly true in cases for patients who are new to Fullscript and who miss the welcome message we send, or patients whose practitioner forgot to actively communicate with them about Fullscript.



Dr. Zaius has created a recommendation for you on Fullscript. You can login to view the recommendation here:

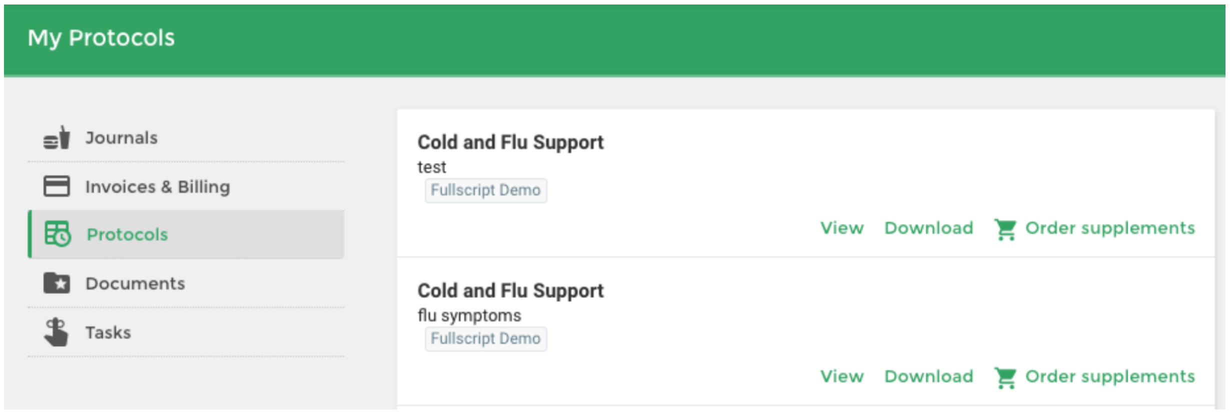
https://us.fullscript.com/welcome/htest/d4015c9f5b/text_rx?src=text

To ensure Fullscript has the patient's mobile information, include it when requesting the Fullscript treatment plan redirect link or update the Fullscript patient record via the patient update endpoint.

Tip
Mobile messages are only supported in countries where Fullscript operates. This is in the US and Canada. Ensure that phone numbers are properly prefixed by the country code (+1 for the US and Canada) and area code. Formatting example: `+12223334444`.

Treatment plan details and order link

If there are active treatment plans for the patient, you can display them in your patient portal. Add the treatment plan `invitation_url` to redirect the patient to their Fullscript account for checkout.

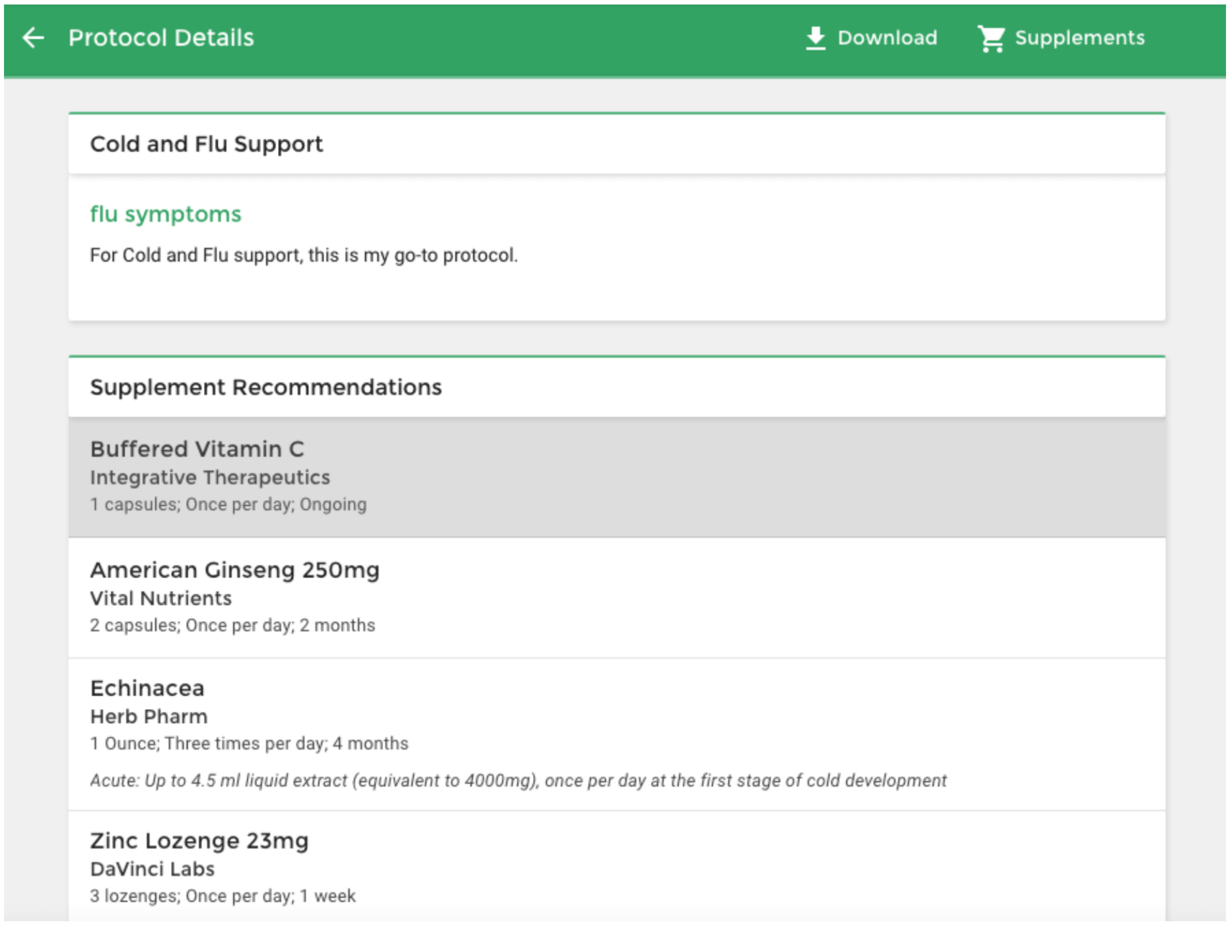


To retrieve the unique patient-facing link, request details for the patient's treatment plan. One of the fields we send back is the invitation_url. Just hook that redirect up to your button and you're good to go!

FYI

The `invitation_url` retrieved from the treatment plan endpoint is the exact same link Fullscript sends the patient via email.

Your patient portal can also include a button to show the details for any of their treatment plans, giving them a quick way to check instructions from their practitioner. Find the treatment plan details with the retrieve a treatment plan endpoint.



Cold and Flu Support

flu symptoms

For Cold and Flu support, this is my go-to protocol.

Supplement Recommendations

Buffered Vitamin C
Integrative Therapeutics
1 capsules; Once per day; Ongoing

American Ginseng 250mg
Vital Nutrients
2 capsules; Once per day; 2 months

Echinacea
Herb Pharm
1 Ounce; Three times per day; 4 months

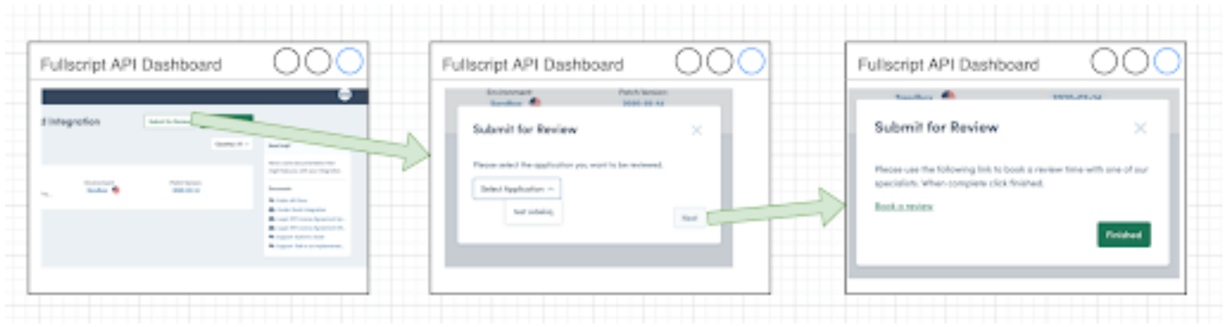
Acute: Up to 4.5 ml liquid extract (equivalent to 4000mg), once per day at the first stage of cold development

Zinc Lozenge 23mg
DaVinci Labs
3 lozenges; Once per day; 1 week

Go live

Before creating production versions of your applications and going live, there is a review and approval process of your sandbox application. This requirement is also reflected in the Fullscript API License Agreement.

To get started, sign in to the Fullscript API Dashboard and click the **Submit for Review** button at the top of the page. In the two popup screens that appear, select the Sandbox application you want reviewed, then use the provided link to book a review time with one of our specialists.



We may have feedback and suggestions for you! Once this process is complete, you'll be allowed to create and configure your production App(s) on the Fullscript API Dashboard. The final go-live process is influenced by contracting and agreed upon marketing efforts. You can request your production application to be listed in our in-app Integrations page via the dashboard also.

Go-Live Development Checklist

Complete development and testing of your sandbox application(s)
Submit your application for review and book a time with one of our specialists
If needed, address feedback or suggestions from the review
Create your production App(s) on the Fullscript API Dashboard

- be sure to provide your production OAuth redirect URI and configure the required OAuth scopes
- **Fullscript Embed:** provide your production Origin URI
- **Webhooks:** provide your app's production endpoint for webhooks

Update your app's production version code to point to production Fullscript API endpoints (US and/or Canada)
Update your app's source to use production OAuth `client ID` and `secret` values, found in your app's OAuth tab on the Fullscript API Dashboard. This value is different for each target environment (US vs Canada).
Fullscript Embed: Update your app's source to use production `public key` value, found in your app's Fullscript Embed tab on Fullscript API Dashboard. This value is different for each target environment (US vs Canada).
Webhooks: Update your app's `secret challenge token` value, and if verifying the validity of the incoming messages, also update the `Fullscript signature secret key`. These values are different for each target environment (US vs Canada).
Go-live 🎉: Request to have your integration listing published on Fullscript's in-app integrations page and arrange co-marketing efforts with the Fullscript team.

What are webhooks?

If you're unfamiliar with webhooks, they are a type of API where we push data to your app instead of you pulling it from Fullscript. To do this, you add a web-accessible endpoint to your app (for example `https://example.com/myapp/fullscript-webhook`) and register that url with us. You tell us which events you want to know about. When any of those events happen, we'll reach out to your endpoint and give you a payload with the data about the event.

So for example, when a practitioner activates a treatment plan, our webhooks notify your app about it. And you get all the details of the new treatment plan.

FYI

Our webhooks include special security features that let your app ascertain a payload is coming from us before acting on it.
Webhook events are sent to your registered application endpoint if and only if:

1. Your application didn't directly cause the event by calling our REST APIs
2. AND, you currently have at least one valid (not revoked) OAuth access token
3. AND, that token has the correct scopes needed for the event that happened

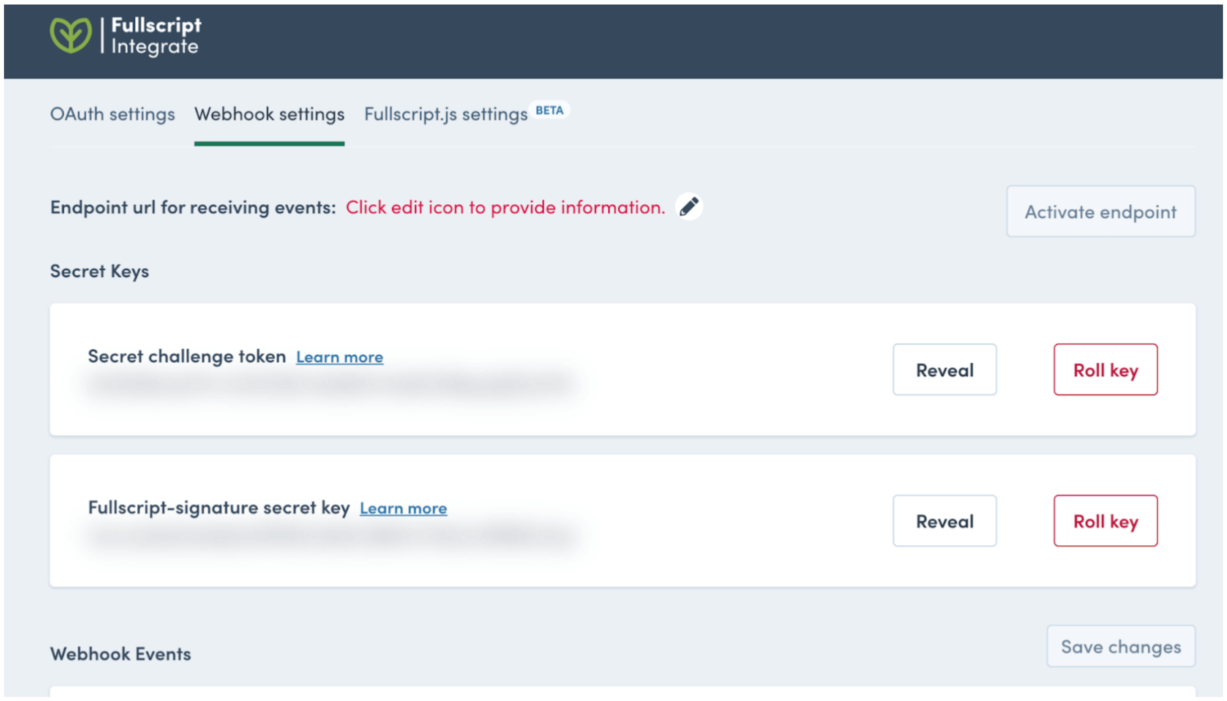
Find more information on webhooks in this how-to guide, and in our reference docs.

Important

It's worth reiterating that webhooks are triggered for actions a practitioner takes through the Fullscript Redirect URL. But they're **not** triggered for anything you do specifically through the REST API. For example, if you use the `patients` endpoint to update the spelling of a Fullscript patient's name, we won't alert you of this since we're pretty sure you already know!

Subscribe to webhooks

Use the Fullscript API Dashboard to subscribe to webhooks. Log in and open the configuration page for your App instance. Open the **Webhook Settings** menu tab to see a page like this:


Fullscript Integrate

OAuth settings | Webhook settings | Fullscript.js settings BETA

Endpoint url for receiving events: Click edit icon to provide information.

Activate endpoint

Secret Keys

Secret challenge token Learn more — Reveal — Roll key

Fullscript-signature secret key Learn more — Reveal — Roll key

Webhook Events — Save changes

Configure your endpoint url

The first thing to do is to add your **Endpoint url for receiving events** by clicking the pencil icon. Your endpoint needs to use HTTPS and have a valid SSL certificate.

Your webhooks endpoint needs to be accessible from outside your organization - so make sure your firewall is set up to permit this. If needed, you can talk with our integration specialists to get a list of IP address(es) you can expect our webhook messages to originate from.

Important

When you click Save Changes, our servers test your endpoint. If it's not accessible, you're unable to proceed. Resolve the issue and try again.

Select events to be notified about

Scroll down the dashboard's **Webhook Settings** page to the list of available webhooks and subscribe to the ones you'd like to use. For example, if you want to be notified when a new treatment plan is created, subscribe to **Treatment plan created**. Be sure to click **Save changes** before you leave the page.

Related config tasks

While you're on this page - copy both your **Secret challenge token** and **Fullscript-signature**

secret key. Save them with other important secret information in your app.

Before you leave the dashboard, make sure that your application is requesting the right set of OAuth scopes (see webhook description) for the webhook events you just subscribed to.

FYI

Adding scopes to your **App** instance in the Dashboard, doesn't update your users' previous authorization tokens with the new scopes. Attempts to use these tokens for activities associated with a new scope will return an Unauthorized error. Whenever you add scopes to your App, we suggest you obtain new tokens by prompting all your users to reauthorize.

Configure your application for webhooks

Confirm receipt

When an event occurs, we start by making a request to your endpoint with an empty body. Configure your application to respond with either an `HTTP 200` or `HTTP 201` response that includes your secret challenge token in the message body. This way, we know it was truly your server that received the response.

```
HTTP 200 OK
Content-type: application/json
{ "challenge": "your-secret-challenge-token" }
```

If we receive a response within 2 seconds, we'll call your endpoint again, but this time we'll include the event payload. Please respond again with your challenge token so we know you've received the event information.

If for some reason your endpoint isn't accessible, we retry the notification up to 6 times with exponential backoff. Each event has its own unique delivery id and event id that you can keep track of. Read more in Missed and repeated webhooks

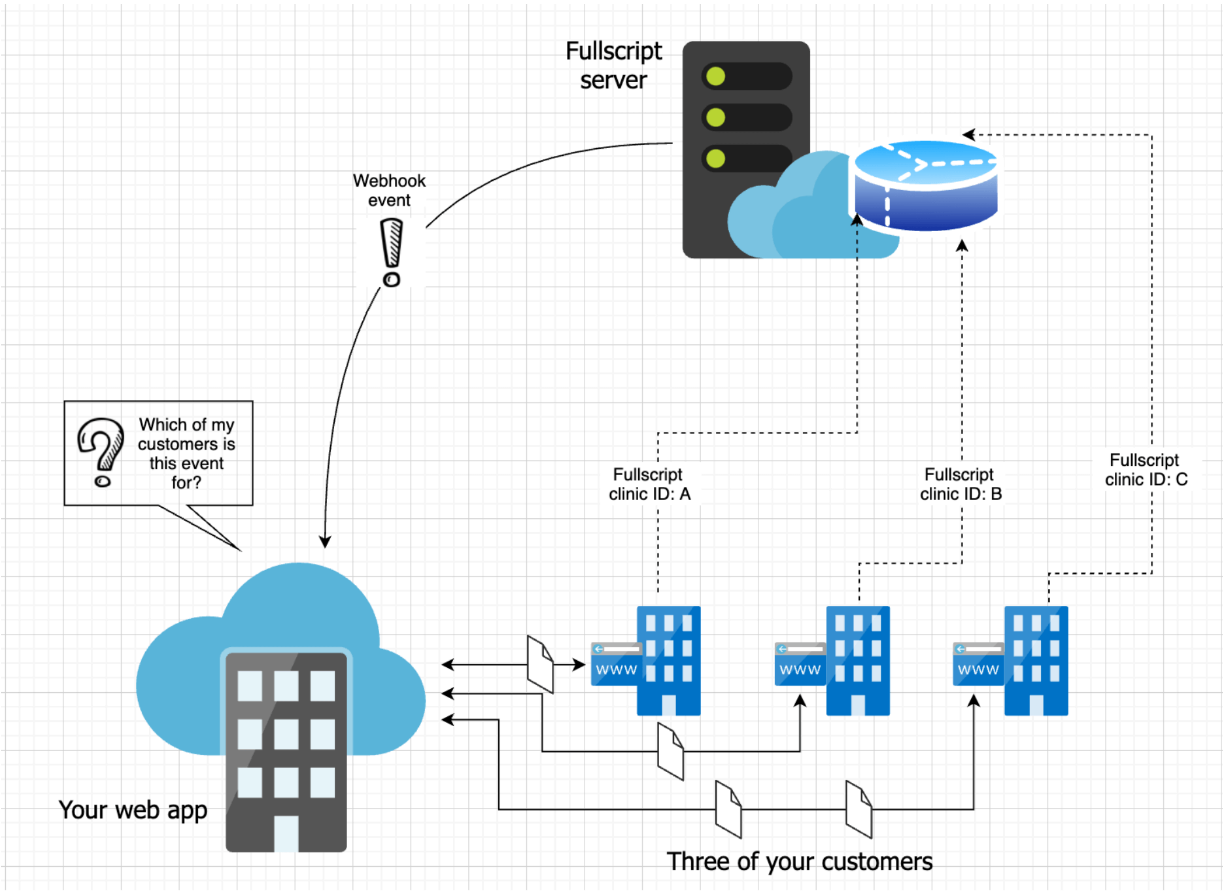
Tip

In the event that your endpoint is down for a while, you can build a manually triggerable action to query our List all webhooks API. You can filter for a specific date range, event id, or for messages that couldn't be delivered. Be aware that only 30 days of history is kept.

Webhook events from multiple clinics

If you’re writing software that will be used by multiple customers, it’s likely that they will each have their own Fullscript clinic.

For example, imagine your software is used by three of your customers, each with their own Fullscript clinic. When you get an event from Fullscript, how will you know which of those customers triggered it? Does it matter?



The answer is yes, it usually matters which of your customers triggered the event. Unless the event is product-related, you need to make sure you’re updating the right customer’s patients and treatment plans. (You may choose to keep a single list of Fullscript products in your database, but elements that are particular to your customers should be stored independently from each other.)

When you receive a webhook event, find the `clinic_id` in the `event_payload`. This identifies the clinic that triggered the event.

```
...
    "event_payload": {
       "event": {
          "id": "x5xxxxx8-92x7-4xxx-9x0x-345x404x94x1",
          "type": "patient.updated",
          "created_at": "2018-10-16T04:00:00.000Z",
          "clinic_id": "xx7x357x-9x36-xxxx-x553-7x3xx398xxx",
          "data": {
...
```

Webhook security

Once you have your basic endpoint and response working, augment it with an additional security step to know that the data payload is secure.

Alter your code to add a signature verification step before responding to the event with `HTTP 200` or `HTTP 201`. The event header includes a timestamp and hash that you can use to

1. verify that the event timestamp is less than a few minutes old
2. compute the same signature using your unique Fullscript-signature secret key

FYI

The Fullscript server will try to send event notices multiple times. So we recommend against adding other complex tasks before returning the 200 or 201 response. If you take too long to respond, your app may receive a second notification for the same event.

Checking the `Fullscript-Signature` header

Fullscript sends a header in the webhook request that can be used in conjunction with your `Webhook Secret Key` to verify the payload. Here is an example of the header:

```
Fullscript-Signature:
t=1591826856,v1=0c262932b0ac6b4952e2fe24fdf419313984a66f6f442e0b8ec4
cb87f2a107ad
```

This represents the format: `t=<timestamp>,v1=<signature>`

The `signature` is generated using a hash-based message authentication code (HMAC) with SHA-256. This hash is generated using 3 things:

- A UTC `timestamp`
- The `request_body` (The POST message's JSON payload string)
- Your `Webhook Secret Key`

The `timestamp` and `request_body` are combined to create the payload provided to the hash function. The `Webhook Secret Key` is used as the key. The same hash can be computed and compared to the `signature` to verify the request.

To recompute and compare the hash, follow these steps

- Extract the `timestamp` and `signature` from the header.
- Create the `payload` by combining the `timestamp` and `request_body` with a single `.` Example: `payload = '1591826856.{"event":{...<more_json>}}'`

- Compute an HMAC with the SHA256 hash function. Use the `Webhook Secret Key` as the key. Use the `payload` string as the message.
- Compare the result with the `signature`.
- If they match, the request payload is legitimate.

FYI

The timestamp is generated directly before we send the payload. It is included in the hash payload in order to help prevent replay attacks. We recommend using this to verify the age of a request with a reasonable tolerance. 5 minutes is usually a good default.

Missed and repeated webhooks

If your registered webhook receiving url isn't accessible, we retry the notification up to 6 times with exponential backoff.

Repeated webhooks

Sometimes, you'll receive more than one notification for the same event. This can happen if:

- your application is performing complicated logic before replying with HTTP 200 or HTTP 201

- your server is slowed by other tasks and can't reply quickly enough

- there's a high volume of traffic that prevents your reply from getting back in time

Each delivery attempt includes the `event_payload`'s event `id`. If desired, you can keep track of the event ids your app has processed:

```
{
   "event_payload": {
     "event": {
       "id": "x5xxxxx8-92x7-4xxx-9x0x-345x404x94x1",
        . . .
}
```

The backoff period for message retries is 1 minute, 5 minutes, 30 minutes, 60 minutes, and 5 hours. If we don't receive your app's acknowledgment of the notification after the last attempt, we stop resending that notice.

Missed webhooks

If you're having trouble receiving webhooks you can check with our deliveries endpoint to see if you missed any webhook events. The payload you get back includes information on how many times we attempted the event delivery as well as any response errors.

In the event that your endpoint is down for a while, you can use our Events API to see which events happened while your system was unavailable, or you can call the List all webhook deliveries endpoint to replay past webhook deliveries.

The deliveries call returns a list of webhooks events that were sent to you. It has a built-in filter for a specific date range, event id, or for messages that couldn't be delivered. Be aware that only 30 days of history is kept, that you can search for ones.

Troubleshooting webhooks

If you're not receiving webhook events or aren't getting notified all the events you expect, here are some things to check:

- Verify that your secret challenge is matches the value in the Fullscript API Dashboard
- Ensure you're subscribed to the events you expect to receive, and that you've saved this configuration with the **Save changes** button on the dashboard.
- Verify that your endpoint uses HTTPS with a valid SSL certificate. Our server verified your endpoint when you entered it. Check that the endpoint is still publicly accessible, and try resetting it from our dashboard to re-trigger the test.
- Check with our deliveries endpoint to see if you missed any webhook events. The payload you get back includes information on how many times we attempted the event delivery as well as any response errors.
- Ensure you have at least valid one user who has successfully authorized your application via OAuth, and their access token is not revoked. Users are able to revoke their token from the Fullscript Web App, so if you've suddenly stopped receiving events, verify that you still have at least one valid token.
- Fullscript only sends the webhook event to your application if your user(s) authorized the correct scopes needed for that event. Check each webhook for a list of scopes needed.
- Are you looking for the right events? While changes made via Fullscript Redirect (new and draft treatment plan links) and Fullscript Embed trigger webhook events,

events **aren't triggered** for any other change your application does through the REST API. Handle changes you initiate without waiting for an event notification.

Order placed

The `order.placed` webhook is triggered when a patient places an order via Fullscript.com.

When a patient purchases a lab, the `order.placed` webhook is triggered. To identify lab orders, inspect `line_items[].lab_type` in the payload. This is the only webhook that signals a lab has been purchased.

Details

Scopes needed:	`patients:order_history`
Event payload:	a subset of the `order object`

Event payload

```
"event_payload": {
   "event": {
     "id": "x5xxxxx8-92x7-4xxx-9x0x-345x404x94x1",
     "type": "order.placed",
     "created_at": "2018-10-16T04:00:00.000Z",
     "clinic_id": "xx7x357x-9x36-xxxx-x553-7x3xx398xxx",
     "data": {
       "id": "391xx4xx-67x5-44x2-8x2x-8886x20x083x",
       "order_number": "R000000000",
       "completed_at": "2018-03-02T13:26:02.000-05:00",
       "treatment_plan_ids": [
         "xx45x0xx-x840-41xx-9922-98xx6001507x"
       ],
       "patient_id": "x1x0196x-5615-4874-xxe4-48x459180x09",
       "line_items": [
         {
           "variant_id": "xx2688x2-4303-4278-xx73-x7083x6146x1",
           "quantity": 3,
           "type": "supplement"
         }
       ]
     }
   }
}
```

Lab order updated

The `lab_order.updated`* webhook event is triggered when:

- A result becomes available (`partial_results`, `results_ready`)
- A previously available result is amended (`results_amended`)

*Available in the US only

It does **not** fire on lab purchases, appointment scheduling, or other state transitions like `purchased`, `processing`, `schedule_appointment`, or `interpretation_shared`.

Status	**Triggers lab_order.updated?**	**Notes**
not_purchased	❌	Internal state only; returned in API
purchased	❌	Use order.placed webhook
schedule_appointment	❌	Only relevant for labs that require phlebotomy
upcoming_appointment	❌	Follows patient scheduling
processing	❌	Sample received but no results yet
partial_results (results_received)	✅	Some results available
results_ready (all_results_received)	✅	All results available
results_amended	✅	One or more results updated

interpretation_shared	❌ (unless amended)	May be reached via auto-share settings

This webhook does **not** trigger on every state change. If your integration depends on showing lab status in real-time, you must poll the lab_order endpoint for state changes outside of result readiness.

Details

Scopes needed:	`patients:order_history`
Event payload:	a subset of the `lab_order` object

Event payload

```
"event_payload": {
   "event": {
     "id": "x5xxxxx8-92x7-4xxx-9x0x-345x404x94x1",
     "type": "lab_order.updated",
     "created_at": "2025-01-01T05:00:00.000Z",
     "clinic_id": "xx7x357x-9x36-xxxx-x553-7x3xx398xxx",
     "data": {
      "lab_order": {
        "id": "x0x50x8x-711x-43xx-85xx-5xx80365xxxx",
        "practitioner_id": "bad7f9c6-dfd5-4227-9611-aac387331801",
        "patient_id": "005aabb2-d97c-4dec-96a6-1e5f33508254",
        "first_name": "John",
        "last_name": "Doe",
        "email": "john.doe@mail.com",
        "status": "results_received"
      }
    }
  }
}
```

Treatment plan created

The `treatment_plan.created` webhook is triggered when a treatment plan is activated. Treatment plans are created in draft mode. A practitioner or staff member activates a treatment plan to share it with the patient.

Details

Scopes needed: `patients:treatment_plan_history`

Event payload: a subset of the `treatment_plan` object

Event payload

```
"event_payload": {
   "event": {
     "id": "x5xxxxx8-92x7-4xxx-9x0x-345x404x94x1",
     "type": "treatment_plan.created",
     "created_at": "2018-10-16T04:00:00.000Z",
     "clinic_id": "xx7x357x-9x36-xxxx-x553-7x3xx398xxx",
     "data": {
         "treatment_plan": {
            "id": "xx45x0xx-x840-41xx-9922-98xx6001507x",
            "active": "true",
            "available_at": "2018-03-02",
            "recommendations": [
               {
               "variant_id": "xx2688x2-4303-4278-xx73-x7083x6146x1",
               "refill": false,
               "units_to_purchase": 10,
               "take_with": "food",
               "dosage": {
                  "amount": "1-2",
                  "frequency": "once per day",
                  "duration": null,
                  "format": "capsule",
                  "additional_info": "with food",
                  "time_of_day": [
                     "morning"
                     ]
                  }
               }
            ],
            "resources": [
               {
                  "id": "7c9e03fd-c41a-45e8-9091-0d420408293b",
                  "attachment_url": "https://api-us-
snd.fullscript.io/api/clinic/resources/7c9e03fd-
c41a-45e8-9091-0d420408293b/download?type=practitioner_resource",
                        "category": "Lifestyle",
                        "name": "logo",
                        "type": "Handout"
                    }
                ],
                "lab_recommendations": [
                    {
                        "id": "233df3fd-c41a-45e8-9091-0d420408293b",
                        "name": "Lab Test 1",
                        "requires_fasting": false,
                        "instructions": "Instructions for Lab Company 1",
                        "tests": [{
                            "id": "5ccf88ae-532e-41c1-9404-1142fbf7c4f7",
                            "name": "Basic Metabolic Panel"
                        }]

                    }
                ]
            }
        }
    }
}
```

Treatment plan updated

The `treatment_plan.updated` webhook is triggered when an active treatment plan's status changes to “cancelled”.

Note: This webhook does not trigger when a draft becomes active, and patient or practitioner changes are not supported on active plans.

Details

Scopes needed:	`patients:treatment_plan_history`
Event payload:	a subset of the `treatment_plan` object

Event payload

```
"event_payload": {
    "event": {
     "id": "x5xxxxx8-92x7-4xxx-9x0x-345x404x94x1",
     "type": "treatment_plan.updated",
     "created_at": "2018-10-16T04:00:00.000Z",
     "clinic_id": "xx7x357x-9x36-xxxx-x553-7x3xx398xxx",
     "data": {
         "treatment_plan": {
            "id": "xx45x0xx-x840-41xx-9922-98xx6001507x",
            "active": "true",
            "available_at": "2018-03-02",
            "recommendations": [
               {
               "variant_id": "xx2688x2-4303-4278-xx73-x7083x6146x1",
               "refill": false,
               "units_to_purchase": 10,
               "take_with": "food",
               "dosage": {
                  "amount": "1-2",
                  "frequency": "once per day",
                  "duration": null,
                  "format": "capsule",
                  "additional_info": "with food",
                  "time_of_day": [
                     "morning"
                     ]
                  }
               }
            ],
            "resources": [
               {
                  "id": "7c9e03fd-c41a-45e8-9091-0d420408293b",
                  "attachment_url": "https://api-us-
snd.fullscript.io/api/clinic/resources/7c9e03fd-
c41a-45e8-9091-0d420408293b/download?type=practitioner_resource",
                  "category": "Lifestyle",
                  "name": "logo",
                  "type": "Handout"
               }
            ],
            "lab_recommendations": [
               {
                  "id": "233df3fd-c41a-45e8-9091-0d420408293b",
                  "name": "Lab Test 1",
                  "requires_fasting": false,
                  "instructions": "Instructions for Lab Company 1",
                  "tests": [{
                     "id": "5ccf88ae-532e-41c1-9404-1142fbf7c4f7",
                     "name": "Basic Metabolic Panel"
```

```
                    }]
                }
            ]
        }
    }
  }
}
```

Treatment plan recommendation updated

The `treatment_plan.recommendation.updated` webhook is triggered when an active treatment plan's details are changed. Includes adding a new variant, removing a variant, or updating a variant (i.e. going from the 60 capsule option to 30 capsule option). Specifically triggered by changes to any of the following attributes of an active treatment plan:

- `variant_id`
- `refill`
- `dosage.amount`
- `dosage.frequency`
- `dosage.duration`
- `dosage.additional_info`
- `dosage.format`
- `quantity_recommended`
- `treatment_plan_id`
- `current_state`
- `units_to_purchase`

Details

Scopes needed: `patients:treatment_plan_history`

Event payload: a subset of the `treatment_plan` object

Event payload

```
"event_payload": {
   "event": {
     "id": "x5xxxxx8-92x7-4xxx-9x0x-345x404x94x1",
     "type": "treatment_plan.recommendation.updated",
     "created_at": "2018-10-16T04:00:00.000Z",
     "clinic_id": "xx7x357x-9x36-xxxx-x553-7x3xx398xxx",
     "data": {
         "treatment_plan": {
            "id": "xx45x0xx-x840-41xx-9922-98xx6001507x",
            "active": "true",
            "available_at": "2018-03-02",
            "recommendations": [
               {
               "variant_id": "xx2688x2-4303-4278-xx73-x7083x6146x1",
               "refill": false,
               "units_to_purchase": 10,
               "take_with": "food",
               "dosage": {
                  "amount": "1-2",
                  "frequency": "once per day",
                  "duration": null,
                  "format": "capsule",
                  "additional_info": "with food",
                  "time_of_day": [
                     "morning"
                     ]
                  }
               }
            ],
            "lab_recommendations": [
               {
                  "id": "233df3fd-c41a-45e8-9091-0d420408293b",
                  "name": "Lab Test 1",
                  "requires_fasting": false,
                  "instructions": "Instructions for Lab Company 1",
                  "tests": [{
                     "id": "5ccf88ae-532e-41c1-9404-1142fbf7c4f7",
                     "name": "Basic Metabolic Panel"
                  }]
               }
            ]
```

```
          }
       }
    }
}
```

Patient created

The `patient.created` webhook is triggered when a new patient is created.

Details

Scopes needed: `patients:read`

Event payload: a subset of the `patients` object

Event payload

```
"event_payload": {
   "event": {
     "id": "x5xxxxx8-92x7-4xxx-9x0x-345x404x94x1",
     "type": "patient.created",
     "created_at": "2018-10-16T04:00:00.000Z",
     "clinic_id": "xx7x357x-9x36-xxxx-x553-7x3xx398xxx",
     "data": {
         "patient": {
            "id": "x1x0196x-5615-4874-xxe4-48x459180x09",
            "first_name": "Example",
            "last_name": "Patient",
            "email": "patient@example.com",
            "date_of_birth": null,
            "gender": "female",
            "discount": 0,
            "total_discount": 0,
            "mobile_number": null,
            "text_message_notification": true,
         }
     }
   }
}
```

Patient updated

The `patient.updated` webhook is triggered when a patient is updated. Specifically triggered when any of the following patient attributes are updated (any patient attribute except metadata or id):

- `first_name`
- `last_name`
- `email`
- `date_of_birth`
- `gender`
- `discount`
- `total_discount`
- `mobile_number`
- `text_message_notification`

Details

Scopes needed:	`patients:read`
Event payload:	a subset of the `patients` object

Event payload

```
"event_payload": {
   "event": {
     "id": "x5xxxxx8-92x7-4xxx-9x0x-345x404x94x1",
     "type": "patient.updated",
     "created_at": "2018-10-16T04:00:00.000Z",
     "clinic_id": "xx7x357x-9x36-xxxx-x553-7x3xx398xxx",
     "data": {
         "patient": {
            "id": "x1x0196x-5615-4874-xxe4-48x459180x09",
            "first_name": "Example",
            "last_name": "Patient",
            "email": "patient@example.com",
            "date_of_birth": null,
            "gender": "female",
            "discount": 0,
            "total_discount": 0,
            "mobile_number": null,
            "text_message_notification": true,
         }
      }
   }
}
```

Patient emancipated

The `patient.emancipated` webhook is triggered when a dependent patient completes the emancipation process.

Details

Scopes needed:	`patients:read`
Event payload:	a subset of the [`patients` object]

Event payload

```
"event_payload": {
   "event": {
     "id": "x5xxxxx8-92x7-4xxx-9x0x-345x404x94x1",
     "type": "patient.emancipated",
     "created_at": "2025-04-16T04:00:00.000Z",
     "clinic_id": "xx7x357x-9x36-xxxx-x553-7x3xx398xxx",
     "data": {
         "patient": {
            "id": "x1x0196x-5615-4874-xxe4-48x459180x09",
            "first_name": "Example",
            "last_name": "Patient",
            "email": "patient@example.com",
            "date_of_birth": null,
            "gender": "female",
            "discount": 0,
            "total_discount": 0,
            "mobile_number": null,
            "text_message_notification": true,
            "is_dependent": false,
            "guardian_patient_id": null
         }
      }
   }
}
```

Product created

The `product.created` webhook is triggered when a new product is added to the Fullscript catalog.

Details

Scopes needed: `catalog:read`

Event payload: a subset of the product object

Event payload

```
"event_payload": {
   "event": {
     "id": "x5xxxxx8-92x7-4xxx-9x0x-345x404x94x1",
     "type": "product.created",
     "created_at": "2018-10-16T04:00:00.000Z",
     "data": {
         "name": "Vitamin C",
         "description_html": "Easy to swallow vitamin c capsules.",
         "dosage": {
            "recommended_amount": "10-60",
            "recommended_frequency": "four times per day",
            "recommended_duration": "as needed",
            "format": "drop",
            "additional_info": "with meals"
         },
         "brand": {
            "id": "x0x50x8x-711x-43xx-85xx-5xx80365xxxx",
            "name": "BioBrand",
            "prefix": "BIO"
         },
         "variants": [
         {
            "id": "xx2688x2-4303-4278-xx73-x7083x6146x1",
            "sku": "20xx9193488",
            "primary": true,
            "units": null,
            "unit_of_measure": null,
            "availability": "In Stock",
            "status": "available",
            "upc": null,
            "msrp": "19.99",
            "supplier_sku": null
         }
         ]
      }
   }
}
```

Product description updated

The `product.description.updated` webhook is triggered when the description of a product in the Fullscript catalog changes.

Details

Scopes needed:	`catalog:read`
Event payload:	a subset of the product object

Event payload

```
"event_payload": {
   "event": {
     "id": "x5xxxxx8-92x7-4xxx-9x0x-345x404x94x1",
     "type": "product.description.updated",
     "created_at": "2018-10-16T04:00:00.000Z",
     "clinic_id": "xx7x357x-9x36-xxxx-x553-7x3xx398xxx",
     "data": {
         "product": {
            "name": "Vitamin C",
            "description_html": "Easy to swallow vitamin c
capsules.",
            "brand": {
               "id": "x0x50x8x-711x-43xx-85xx-5xx80365xxxx",
               "name": "BioBrand",
               "prefix": "BIO"
            },
            "variants": [
            {
               "id": "xx2688x2-4303-4278-xx73-x7083x6146x1",
               "sku": "20xx9193488",
               "primary": true,
               "units": null,
               "unit_of_measure": null,
               "availability": "In Stock",
               "status": "available",
               "upc": null,
               "msrp": "19.99",
               "supplier_sku": null
            }
            ],
            "dosage": {
               "recommended_amount": "10-60",
               "recommended_frequency": "four times per day",
               "recommended_duration": "as needed",
               "format": "drop",
               "additional_info": "with meals"
            },
         }
      }
   }
}
```

Product updated

The `product.updated` webhook is triggered when a product in the Fullscript catalog is updated. Specifically, when changes are made to the following attributes (note that a change to the description triggers `product.description.updated` instead):

- `name`
- `dosage`
- `recommended_amount`
- `recommended_frequency`
- `recommended_duration`
- `format`
- `additional_info`

Details

Scopes needed: `catalog:read`

Event payload: a subset of the `product` object

Event payload

```
"event_payload": {
   "event": {
     "id": "x5xxxxx8-92x7-4xxx-9x0x-345x404x94x1",
     "type": "product.updated",
     "created_at": "2018-10-16T04:00:00.000Z",
     "clinic_id": "xx7x357x-9x36-xxxx-x553-7x3xx398xxx",
     "data": {
         "product": {
            "name": "Vitamin C",
            "description_html": "Easy to swallow vitamin c
capsules.",
            "brand": {
               "id": "x0x50x8x-711x-43xx-85xx-5xx80365xxxx",
               "name": "BioBrand",
               "prefix": "BIO"
            },
            "variants": [
            {
               "id": "xx2688x2-4303-4278-xx73-x7083x6146x1",
               "sku": "20xx9193488",
               "primary": true,
               "units": null,
               "unit_of_measure": null,
               "availability": "In Stock",
               "status": "available",
               "upc": null,
               "msrp": "19.99",
               "supplier_sku": null
            }
            ],
            "dosage": {
               "recommended_amount": "10-60",
               "recommended_frequency": "four times per day",
               "recommended_duration": "as needed",
               "format": "drop",
               "additional_info": "with meals"
            },
         }
      }
   }
}
```

Product updated

The `product.updated` webhook is triggered when a product in the Fullscript catalog is updated. Specifically, when changes are made to the following attributes (note that a change to the description triggers `product.description.updated` instead):

- `name`
- `dosage`
- `recommended_amount`
- `recommended_frequency`
- `recommended_duration`
- `format`
- `additional_info`

Details

Scopes needed: `catalog:read`

Event payload: a subset of the `product` object

Event payload

```
"event_payload": {
   "event": {
     "id": "x5xxxxx8-92x7-4xxx-9x0x-345x404x94x1",
     "type": "product.updated",
     "created_at": "2018-10-16T04:00:00.000Z",
     "clinic_id": "xx7x357x-9x36-xxxx-x553-7x3xx398xxx",
     "data": {
         "product": {
            "name": "Vitamin C",
            "description_html": "Easy to swallow vitamin c
capsules.",
            "brand": {
               "id": "x0x50x8x-711x-43xx-85xx-5xx80365xxxx",
               "name": "BioBrand",
               "prefix": "BIO"
            },
            "variants": [
            {
               "id": "xx2688x2-4303-4278-xx73-x7083x6146x1",
               "sku": "20xx9193488",
               "primary": true,
               "units": null,
               "unit_of_measure": null,
               "availability": "In Stock",
               "status": "available",
               "upc": null,
               "msrp": "19.99",
               "supplier_sku": null
            }
            ],
            "dosage": {
               "recommended_amount": "10-60",
               "recommended_frequency": "four times per day",
               "recommended_duration": "as needed",
               "format": "drop",
               "additional_info": "with meals"
            },
         }
      }
   }
}
```

Events overview

The Fullscript events endpoint that lets you request a list of recent events and pull details for each. It's the functional opposite of our webhook system, which pushes notifications to you.

There's no subscription process for events. Your app requests events on-demand or at regularly scheduled intervals.

You have access to events for the last 30 days (filterable by day) in the following categories, each with their own endpoint inside /events:

- patient events
- treatment plan events
- order events
- product events

FYI

Events **are not created** for changes made through our REST API (except those made via the draft/new treatment plan redirect links or through Fullscript Embed). We assume that since you initiated the event, you already know about it!
When you call any of the event endpoints, you get a list of applicable event `ids`. Use retrieve an event to request the event details, including the `clinic_id` for the clinic that caused this event.

Tip

Consider keeping a list of events your app processed in the last 30 days. This ensures you don't waste time on an event you've already handled.

Event access

To access the event list and event details calls, your app needs to include the OAuth bearer token of a user who has successfully authorized your application for the relevant scopes. Do not use a revoked or expired token.

```
curl "https://api-us-snd.fullscript.io/api/events/patients" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Tip

If you want to check for events at several points during the day, you may need a keep-alive mechanism for at least one user token.

The bearer token you use is affects which events returned.

Important

Only the events that happen at the clinic the user is registered with can be accessed with their token.

If your app only supports one Fullscript clinic, there’s nothing else you need to do. But if you plan to support multiple clinics, you need to call the endpoint on behalf of a user of each configured clinic.

Imagine your app is a health portal for physiotherapy centers. If you only support one center, all your physios, staff, and patients are part of the same center. No matter whose OAuth token you use, you receive events for the Fullscript clinic associated with that center.

Let’s imagine this physiotherapist center was in Washington, but the owner has a friend with a center in New York. The New York center also uses Fullscript, but they have their own Fullscript clinic. No doubt there are several changes you need to make to your software to support multiple physio centers. Among these is a change to make your app poll the Fullscript events endpoint on behalf of each Fullscript clinic.

```
centers.forEach(center => {
  // check for new patient orders at this center, use a bearer token
for a user from the current center
});
```

In our example where you support a center in Washington and a separate one in New York (each with its own Fullscript clinic), you need to call the events endpoint once with a bearer token for a user in Washington, and once with a token for a user in New York.

Note that the clinic is specified implicitly, through the bearer token.

Fullscript Labs Integration Guide

Overview

Lab recommendations originate within a treatment plan and, when purchased by a patient, become lab orders. These orders contain one or more **lab tests**, each of which can produce individual **results**.

Integrators must:

- Create and activate a treatment plan with lab recommendations.
- Monitor purchase and fulfillment via webhook events.
- Retrieve lab results using the API.

Lab Order Lifecycle

The lab order progresses through several statuses:

Status	Description	Triggers lab_order.updated ?
not_purchased	Lab recommended, but not purchased yet	❌
purchased	Patient has purchased the lab	❌ (*see order.placed)
schedule_appointment	The lab purchased requires an appointment to be scheduled	❌
upcoming_appointment	Appointment scheduled	❌
processing	Sample received and is being processed	❌
partial_results (results_received)	Some but not all results available	✅
results_ready (all_results_received)	All results available	✅
results_amended	Previously published results have been updated	✅

interpretation_ready	The results are ready to be interpreted in Fullscript	❌
interpretation_shared	Practitioner shared interpretation with patient	❌

Not all labs require appointments. `schedule_appointment` and `upcoming_appointment` only apply to labs that do.

Entities & Relationships

- **Treatment Plan** – Includes one or more lab recommendations.
- **Lab Order** – Created once the lab recommendation is purchased.
- **Lab Test** – A component of the lab order; there may be multiple.
- **Lab Result** – Associated with a test(s).

Mapping Tests and Results

The `lab_order` object contains two key arrays:

- `tests[]`: Lists all tests ordered.
- `results[]`: Lists results received, which may or may not directly align one-to-one with the tests.

Important Notes:

- There might be no direct mapping between result entries and test entries.
- This reflects how lab providers deliver data. Some group multiple tests into one result; others separate them.

Current Workaround:

- If only **one test** exists in the lab order, label the result using the test name.
- If **multiple tests** exist, fallback logic is recommended:
  - Use the lab order name or
  - Use the result ID as a filename.

API Reference

Retrieve All Lab Orders

```
GET /api/clinic/labs/orders
```

Returns a list of lab orders for the clinic.

Retrieve a Single Lab Order

```
GET /api/clinic/labs/orders/{id}
```

Returns detailed info on a lab order, including:

- `state`
- `tests[]`
- `results[]`

Retrieve a Lab Test

```
GET /api/clinic/labs/tests/{id}
```

Returns metadata about an individual lab test.

Treatment Plan API (for creating Lab Recommendations)

Labs are added to treatment plans under the `lab_recommendations` array.

Once a draft treatment plan is created, you must activate it:

```
PATCH /api/clinic/treatment_plans/{treatment_plan_id}/activate
```

This will make the plan visible to the patient and enable lab purchase.

Webhooks

order.placed

Triggered when the patient purchases the lab.

```
{
  "type": "order.placed",
  "data": {
    "line_items": [
      {
        "variant_id": "...",
        "type": "labs" // Identify lab orders
      }
    ]
  }
}
```

Use this to detect lab purchases.

lab_order.updated

Triggered only when **results** become available or are amended. Does **not** trigger on purchase or appointment scheduling.

```
{
  "type": "lab_order.updated",
  "data": {
    "lab_order": {
      "id": "...",
      "status": "all_results_received" | "results_received" |
"results_amended"
    }
  }
}
```

Note: `all_results_received` maps to the `results_ready` state and `results_received` maps to the `partial_results` state on lab orders.

You must **poll the API** to track state changes like `purchased`, `processing`, and `appointment` statuses.

How to test labs in the Fullscript sandbox

Use this step-by-step guide to simulate the lab ordering flow within your EHR integration. This ensures your integration handles lab purchases, status transitions, and webhook behavior correctly.

1. Navigate to: https://us-snd.fullscript.io/login
2. Log in or create a test practitioner account
    - Sign up as a practitioner on Fullscript if you haven't already. You must enter license information to proceed with lab orders.
3. Add your NPI number (if you skipped it during onboarding)
    - Navigate to **Settings → Account → Basic info**
    - Click **Edit license information**
    - Enter the test NPI number: `1111111111`
    - Click **Save license information**
4. Navigate to Labs
    - Go to **Catalog → Labs**
    - You will see the **Register for labs** button.
5. *Note: If the `Register for labs` button is greyed out, **hover over it** to view the tooltip. It will explain what's missing and include a hyperlink to the relevant page to enter your info.*
6. Complete the lab registration process
    - Click the Register for labs button.
    - Re-enter your first and last name (NPI should be pre-filled)
    - Leave license info blank (optional)
    - Click the Next button.
7. Agree to the legal documents
    - Review and agree to Fullscript Labs' Terms of Service. Click the Next button.
    - Review and electronically sign the BAA. Click the Submit button.
    - Agree to the authorization network. Click the Done button.
8. Create and send a lab treatment plan
    - Go to **Patients**, then create a new test patient (if needed).

- Create a treatment plan for that patient.
- Add one or more labs from the **Labs catalog**.
- Send the treatment plan to the patient.

9 Log in as the test patient and order the lab

- Log in using the test patient's email
- Go to **My Health** → **Plans** to view the recommended labs
- Complete any required personal information if prompted
- Add the lab(s) to the cart and proceed to checkout

10 Use the sandbox test payment method

- Card number: `4242 4242 4242 4242`
- Expiration: any future date
- CVC: any 3 digits
- Click **Place your order**

EHR status confirmation

After a lab order is placed, you may observe several status transitions depending on how your system is configured. The following intermediate statuses are optional to surface in your EHR, but are available for testing:

- `purchased` – Indicates the patient has successfully purchased the lab.
- `schedule_appointment` – Appears when the lab requires the patient to book an appointment (e.g., for phlebotomy).
- `upcoming_appointment` – Indicates that the appointment has been scheduled by the patient.
- `results_ready`(`all_results_received`) – All results have been received and are available for the practitioner. (Note: This must be manually triggered by Fullscript in sandbox—see below.)
- `interpretation_shared` – Indicates that the results have been shared with the patient.
    - If the practitioner has **"Automatically share Quest Diagnostics results"** enabled under **Settings** → **Labs**, the status will automatically transition

from `results_ready` to `interpretation_shared` once all results are available.

- If this setting is **disabled**, the lab order will remain in `results_ready` until the practitioner **manually** chooses to share the results from within Fullscript. Only then will the status update to `interpretation_shared`.

Simulating `all_results_received` status

To test the delivery of lab results and trigger the lab_order.updated webhook:

1. Complete the lab order flow in sandbox.
2. Email/Slack us the following:
   - Practitioner name
   - Test patient name
3. Fullscript will push a test result file to your EHR.
4. You should then receive the `lab_order.updated` webhook with the `all_results_received` status.

TECHNICAL REFERENCE:

Pagination

All top-level list endpoints can be paginated. They return a `meta` tag that contains all pagination data.

Example Request

```
curl "https://api-us-snd.fullscript.io/api/clinic/
patients?page[number]=2&page[size]=20" \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Example Response

```
{
  "patients": [
    {...}
  ],
  "meta": {
    "current_page": 2,
    "next_page": null,
    "prev_page": 1,
    "total_pages": 2,
    "total_count": 40
  }
}
```

The maximum page size is 100.

Request IDs

All API requests sent will respond with an **X-Request-Id** in its response header. This value uniquely identifies each request. If you are having trouble with a particular request please provide the **X-Request-Id** identifier for the fastest possible resolution.

Example

```
'x-request-id: 06ded6a2-cb35-43e7-9f83-a4b705e3e588'
```

Versioning

The API version you use controls the behaviour of the API. The first time you register to use Fullscript's API you will be assigned the latest version.

The current version for Fullscript's API is Version 2. If we ever make a breaking change to the API we will release a new dated `patch_version` with the new behaviour. We do this to avoid breaking any of your code. It's up to you to upgrade if / when you want to.

The latest `patch_version` is `2020-02-14`.

Backwards compatible changes

The Fullscript API considers the following changes to be backwards compatible (ie. **non-breaking** changes).

- Adding new API resources
- Adding new optional request parameters to an existing resource
- Adding new keys/attributes to existing API responses
- Changing the order of parameters in an existing API response
- Changing the format or length of IDs or strings

Rate Limits

The Fullscript API has safeguards to limit the number of requests that can be made in a given timeframe. This helps maximize the API's stability and reliability when dealing with bursts of incoming traffic.

The current rate limit (across all API endpoints) is **5000 requests in a rolling 5m period**.

Applications that send many requests in quick succession may see error responses.

Clinic

This endpoints retrieves information about the clinic provided from the `X-FS-Clinic-Key`.

Retrieve a clinic

This endpoints retrieves information about the clinic provided from the `X-FS-Clinic-Key`.

Arguments

No arguments...

GET

/api/clinic

```
curl "https://api-us-snd.fullscript.io/api/clinic" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

200
OK

403
Forbidden

Staff

The `staff` object contains a record of all available staff members that belong to a clinic. The API allows you to create and update staff members. It also allows you to list all staff and find individual staff members.

List all staff

This resource allows you to list all of a clinic's staff.

Arguments

No arguments...

GET

/api/clinic/staff

```
curl "https://api-us-snd.fullscript.io/api/clinic/
staff" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

200
OK

403
Forbidden

Create a staff member

Creates a new staff member

Arguments

`email`
string
REQUIRED
Unique Email for the staff member

`first_name`
string
REQUIRED
Staff member's first name

`last_name`
string
REQUIRED
Staff member's last name

`metadata`
object
Metadata to be attached to the staff member.
Show child attributes

POST
/api/clinic/staff

```
curl -X 'POST' "https://api-us-snd.fullscript.io/api/
clinic/staff" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX' \
  -d $'{
  "email": "string",
  "first_name": "string",
  "last_name": "string",
  "metadata": {
    "id": "string"
  }
}'
```

Responses

201
Created

403
Forbidden

422
Unprocessable Entity

Retrieve a staff member

Retrieves an existing staff member. You need to supply the unique ID for the staff member.

Arguments

`id`
string
REQUIRED
Unique ID for the staff member

GET
/api/clinic/staff/{id}

```
curl "https://api-us-snd.fullscript.io/api/clinic/
staff/{id}" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

200
OK

403
Forbidden

404
Not Found

Update a staff member

Updates a specific staff member with the attributes that you pass in. Parameters not provided will remain unchanged.

Arguments

`id`
string
REQUIRED
Unique Identifier for the staff member

`email`
string
Unique Email for the staff member

`first_name`
string
Staff member's first name

`last_name`
string
Staff member's last name

`metadata`
object
Metadata to be attached to the staff member.
Show child attributes

PATCH
/api/clinic/staff/{id}

```
curl -X 'PATCH' "https://api-us-snd.fullscript.io/api/
clinic/staff/{id}" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX' \
  -d $'{
  "email": "string",
  "first_name": "string",
  "last_name": "string",
  "metadata": {
    "id": "string"
  }
}'
```

Responses

200

OK

403

Forbidden

404

Not Found

422

Unprocessable Entity

Practitioners

The `practitioner` object contains a record of all available practitioners that belong to a clinic. The API allows you to create and update practitioners. It also allows you to list all practitioners and find individual practitioners.

List all practitioners

This resource allows you to list all of a clinic's practitioners.

Arguments

No arguments...

GET

/api/clinic/practitioners

```
curl "https://api-us-snd.fullscript.io/api/clinic/
practitioners" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

200

OK

403
Forbidden

Create a practitioner

Creates a new practitioner

Arguments

`practitioner_type_id`
string
REQUIRED
Unique Identifier for the Practitioner's Type

`email`
string
REQUIRED
Unique Email for the Practitioner

`first_name`
string
REQUIRED
Practitioner's first name

`last_name`
string
REQUIRED
Practitioner's last name

`metadata`
object
Metadata to be attached to the practitioner.
Show child attributes

POST
/api/clinic/practitioners

```
curl -X 'POST' "https://api-us-snd.fullscript.io/api/
clinic/practitioners" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX' \
  -d $'{
  "practitioner_type_id": "string",
  "email": "string",
  "first_name": "string",
  "last_name": "string",
  "metadata": {
    "id": "string"
  }
}'
```

Responses

201
Created

403
Forbidden

404
Not Found

Retrieve a practitioner

Retrieves an existing practitioner. You need to supply the unique ID for the practitioner.

Arguments

`id`
string
REQUIRED
Unique ID for the Practitioner

GET
/api/clinic/practitioners/{id}

```
curl "https://api-us-snd.fullscript.io/api/clinic/
practitioners/{id}" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

200
OK

403
Forbidden

404
Not Found

Update a practitioner

Updates a specific practitioner with the attributes that you pass in. Parameters not provided will remain unchanged.

Arguments

`id`
string
REQUIRED
Unique Identifier for the Practitioner

`email`
string
Unique Email for the Practitioner

`first_name`
string
Practitioner's first name

`last_name`
string
Practitioner's last name

`metadata`
object
Metadata to be attached to the practitioner.
Show child attributes

PATCH
/api/clinic/practitioners/{id}

```
curl -X 'PATCH' "https://api-us-snd.fullscript.io/api/
clinic/practitioners/{id}" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX' \
  -d $'{
  "email": "string",
  "first_name": "string",
  "last_name": "string",
  "metadata": {
    "id": "string"
  }
}'
```

Responses

200
OK

403
Forbidden

404
Not Found

Practitioner Types

The `practitioner type` object contains details for all the practitioner types available. The API allows you to list all practitioner types.

List all practitioner types

This resource allows you to list all practitioner types.

Arguments

No arguments...

GET
/api/catalog/practitioner_types

```
curl "https://api-us-snd.fullscript.io/api/catalog/
practitioner_types" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

200

OK

403
Forbidden

Protocols

The `protocol` object contains a record of all treatment plan protocols made by, shared with a practitioner or created by Fullscript. Protocols use a treatment plan as a template that can be re-used when writing prescriptions. Protocols were previously known as templates.

List all Fullscript protocols

The Fullscript protocol object contains a list of all treatment plan protocols that have been created by Fullscript.

Arguments

`sort_by`
string
Accepts a `name` argument.

`order_by`
string
Can take an argument of `ASC` or `DESC`.

GET
/api/clinic/fullscript_protocols

```
curl "https://api-us-snd.fullscript.io/api/clinic/
fullscript_protocols" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

200
OK

List all practitioner protocols

The practitioner protocol object contains a record of all treatment plan protocols that belong to a practitioner.

Arguments

`practitioner_id`
string
REQUIRED
Unique ID for the Practitioner

`current_state`
string
Takes an argument of `active`, `draft` or `cancelled` to return `active`, `draft` or `cancelled` protocols.

`sort_by`
string
Accepts one of the following arguments: `name` or `current_state`.

`order_by`
string
Ordering defaults to `ASC` and can take an argument of `ASC` or `DESC`.

`ownership_type`
string
Takes an argument of `owned`, `shared` or `all` to return only protocols created by the practitioner, only protocols shared with the practitioner, or both protocols created by and shared with the practitioner, respectively. If this argument is not included, only protocols created by the practitioner are returned.

GET
/api/clinic/practitioners/{practitioner_id}/protocols

```
curl "https://api-us-snd.fullscript.io/api/clinic/
practitioners/{practitioner_id}/protocols" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

200
OK

403
Forbidden

404
Not Found

422

Unprocessable Entity

Retrieve a protocol

Retrieves an existing protocol.

Arguments

`id`
string
REQUIRED
Unique identifier of the protocol.

GET
/api/clinic/protocols/{id}

```
curl "https://api-us-snd.fullscript.io/api/clinic/
protocols/{id}" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

200
OK

403
Forbidden

404
Not Found

Templates

The `template` object contains a record of all treatment plan templates made by a practitioner. Templates use a treatment plan as a template that can be re-used when writing prescriptions. Templates are soon being deprecated and replaced with protocols.

List all practitioner templates

The practitioner template object contains a record of all treatment plan templates that belong to a practitioner.

Arguments

`practitioner_id`
string
REQUIRED
Unique ID for the Practitioner
GET
/api/clinic/practitioners/{practitioner_id}/templates

```
curl "https://api-us-snd.fullscript.io/api/clinic/
practitioners/{practitioner_id}/templates" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

200
OK

403
Forbidden

404
Not Found

Retrieve a template

Retrieves an existing template.

Arguments

`id`
string
REQUIRED
Unique identifier of the Template.
GET
/api/clinic/templates/{id}

```
curl "https://api-us-snd.fullscript.io/api/clinic/
templates/{id}" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

200
OK

403
Forbidden

404
Not Found

Labs

The `Lab` object contains a record of all ordered labs for a patient. Labs begin as recommendations within a treatment plan. Once purchased by the patient, they become lab orders. Lab orders can contain one or more tests and return one or more results.

New to Labs? Start with the Labs integration guide for an overview of lab status transitions, webhook behavior, and result mapping best practices.

List all lab orders

Lists all ordered labs.

Arguments

`patient_id`
string
Filter orders by `patient_id`.

GET
/api/clinic/labs/orders

```
curl "https://api-us-snd.fullscript.io/api/clinic/labs/
orders" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

200
OK

403
Forbidden

404
Not Found

Retrieve a lab order

Retrieves a ordered lab.

Arguments

`id`
string
REQUIRED
Unique identifier of the lab order.

GET
/api/clinic/labs/orders/{id}

```
curl "https://api-us-snd.fullscript.io/api/clinic/labs/
orders/{id}" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

200
OK

404
Not Found

Retrieve a lab test

Retrieves a lab test.

Note: Lab tests are returned separately from lab results. There is currently no guaranteed one-to-one mapping between `results[]` and `tests[]` in the lab order object. See the Labs Integration Guide for recommendations.

Arguments

`id`
string
REQUIRED
Unique identifier of the lab test.

GET
/api/clinic/labs/tests/{id}

```
curl "https://api-us-snd.fullscript.io/api/clinic/labs/
tests/{id}" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

200
OK

404
Not Found

Resources

The `resource` object contains a record of all resources made by or shared with a practitioner. The API allows you to create, update, and remove resources. It also allows you to retrieve resources linked to a treatment plan.

List all practitioner resources

Lists all Resources that the Practitioner has access to.

Arguments

`practitioner_id`
string
REQUIRED
Unique ID for the Practitioner

GET

/api/clinic/practitioners/{practitioner_id}/resources

```
curl "https://api-us-snd.fullscript.io/api/clinic/
practitioners/{practitioner_id}/resources" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

200
OK

Create a resource

Creates a new Resource for the Practitioner. Only the following file types are allowed: `jpg`, `jpeg`, `png`, `pdf`, `doc`, `xlsx`, `pptx`.

Arguments

`practitioner_id`
string
REQUIRED
The unique ID of the practitioner.

`type`
string
The type of resource. Can be any of the following strings: `Guide`, `Handout`, `Infographic`, `Meal plan`, or `Workbook`.

`category`
string
The category of the resource. Can be any of the following strings: `Nutrition`, `Lifestyle`, `Physical activity`, `Supplements`, or `Health conditions`.

`attachment`
string
REQUIRED
The resource that is being uploaded.

`name`
string
REQUIRED
The name of the resource that is being uploaded.

`treatment_plan_id`
string
The treatment plan ID with which the resource is being associated.

`save_to_library`

string
REQUIRED
A boolean value that signifies whether the resource should be saved to the practitioner's resource library.
POST
/api/clinic/practitioners/{practitioner_id}/resources

```
curl -X 'POST' "https://api-us-snd.fullscript.io/api/
clinic/practitioners/{practitioner_id}/resources" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX' \
  -d $'{
  "type": "string",
  "category": "string",
  "attachment": "string",
  "name": "string",
  "treatment_plan_id": "string",
  "save_to_library": "string"
}'
```

Responses

200
OK

404
Not Found

422
Unprocessable Entity

Retrieve a practitioner's resource

Retrieves a Resource that the Practitioner has access to.

Arguments

`practitioner_id`
string
REQUIRED
Unique ID for the Practitioner.
`id`

string
REQUIRED
Unique ID for the Resource.
GET
/api/clinic/practitioners/{practitioner_id}/resources/{id}

```
curl "https://api-us-snd.fullscript.io/api/clinic/
practitioners/{practitioner_id}/resources/{id}" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

200
OK

403
Forbidden

404
Not Found

Update a resource

Updates a Practitioner's Resource.

Arguments

`practitioner_id`
string
REQUIRED
Unique ID for the Practitioner.
`id`
string
REQUIRED
Unique ID for the Resource.
PUT
/api/clinic/practitioners/{practitioner_id}/resources/{id}

```
curl -X 'PUT' "https://api-us-snd.fullscript.io/api/
clinic/practitioners/{practitioner_id}/resources/{id}"
\
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

200
OK

403
Forbidden

422
Unprocessable Entity

Delete a resource

Deletes an existing Practitioner's Resource.

Arguments

`practitioner_id`
string
REQUIRED
Unique ID for the Practitioner.

`id`
string
REQUIRED
Unique ID for the Resource.

DELETE
/api/clinic/practitioners/{practitioner_id}/resources/{id}

```
curl -X 'DELETE' "https://api-us-snd.fullscript.io/api/
clinic/practitioners/{practitioner_id}/resources/{id}"
\
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

204
No Content
403
Forbidden

404
Not Found

List All Fullscript Resources

The Fullscript resource object contains a list of all resources that have been created by Fullscript.

Arguments

No arguments...

GET
/api/clinic/fullscript_resources

```
curl "https://api-us-snd.fullscript.io/api/clinic/
fullscript_resources" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

200
OK

403
Forbidden

Metadata

Metadata allows you to update specific objects in Fullscript with an ID of your choosing. Currently we support updating Patient, Practitioner, Staff, and TreatmentPlan objects with metadata.

Metadata is useful for storing identifiers from your system into Fullscript — making it easier to link up users from your system into ours. For example, you could use metadata to attach your system's user ID to a Practitioner or Patient. Then using the metadata endpoint you can retrieve that same object using the ID's from your system.

Find objects

To look up objects by your system's identifier. The object type and ID can be provided to filter results.

Arguments

`id`
string
Your system's identifier.

`type`
object
The type of object. Can be one of `patient`, `practitioner`, `staff`, or `treatment_plan`.

GET
/api/clinic/metadata

```
curl "https://api-us-snd.fullscript.io/api/clinic/
metadata" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

200
OK

403
Forbidden

Patients

The `patient` object contains a record of all available patients that belong to a clinic. The API allows you to create and update patients. It also allows you to list all patients and find individual patients.

List all dependents

This resource allows you to retrieve all dependents linked to a guardian patient.

Arguments

`patient_id`

string
REQUIRED
The ID of the guardian patient to list dependents for.
GET
/api/clinic/patients/{patient_id}/dependents

```
curl "https://api-us-snd.fullscript.io/api/clinic/
patients/{patient_id}/dependents" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

200
OK

404
Not Found

List all patients

This resource allows you to list all of a clinic's patients.

Arguments

`patient_type`
string
The type of patient to filter by. Valid options are `all`, `dependent`, or `guardian`. If not provided, all patients will be returned.
GET
/api/clinic/patients

```
curl "https://api-us-snd.fullscript.io/api/clinic/
patients" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

200
OK

403

Forbidden

Create a patient

Creates a new patient

Arguments

`email`
string
REQUIRED
Unique Email for the Patient

`first_name`
string
REQUIRED
Patient's first name

`last_name`
string
REQUIRED
Patient's last name

`date_of_birth`
string
Patient's date of birth in the format `yyyy-mm-dd`

`gender`
string
Biological sex of the patient. Valid options are 'male', 'female', or 'x'.

`mobile_number`
string
Patient's mobile number in the format `+12223334444`

`send_welcome_email`
string
Sends a welcome email to the patient upon successful creation. Defaults to true.

`discount`
string
Patient discount level (in percentage). This discount does not include the clinic discount. Defaults to 0.

`metadata`
object
Metadata to be attached to the patient.
Show child attributes

POST

`/api/clinic/patients`

```
curl -X 'POST' "https://api-us-snd.fullscript.io/api/
clinic/patients" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX' \
  -d $'{
  "email": "string",
  "first_name": "string",
  "last_name": "string",
  "date_of_birth": "string",
  "gender": "string",
  "mobile_number": "string",
  "send_welcome_email": "string",
  "discount": "string",
  "metadata": {
    "id": "string"
  }
}'
```

Responses

201
Created

403
Forbidden

422
Unprocessable Entity

Retrieve a patient

Retrieves an existing patient. You need to supply the unique ID for the patient.

Arguments

`id`
string
REQUIRED
Unique ID for the Patient

GET
/api/clinic/patients/{id}

```
curl "https://api-us-snd.fullscript.io/api/clinic/
patients/{id}" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

200
OK

403
Forbidden

404
Not Found

Update a patient

Updates a specific patient with the attributes that you pass in. Parameters not provided will remain unchanged.

Arguments

`id`
string
REQUIRED
Unique ID for the Patient

`email`
string
Unique Email for the Patient

`first_name`
string
Patient's first name

`last_name`
string
Patient's last name

`date_of_birth`
string
Patient's date of birth in the format `yyyy-mm-dd`

`gender`

string
Biological sex of the patient. Valid options are 'male', 'female', or 'x'.

`mobile_number`
string
Patient's mobile number in the format `+12223334444`

`discount`
string
Patient discount level (in percentage). This discount does not include the clinic discount. Defaults to 0.

`metadata`
object
Metadata to be attached to the patient.
Show child attributes

PATCH
/api/clinic/patients/{id}

```
curl -X 'PATCH' "https://api-us-snd.fullscript.io/api/
clinic/patients/{id}" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX' \
  -d $'{
  "email": "string",
  "first_name": "string",
  "last_name": "string",
  "date_of_birth": "string",
  "gender": "string",
  "mobile_number": "string",
  "discount": "string",
  "metadata": {
    "id": "string"
  }
}'
```

Responses

200
OK

403

Forbidden

404
Not Found

422
Unprocessable Entity

Search for patients

This resource allows you to search for patients by `first_name`, `last_name`, or by `email`.

Arguments

`query`
string
Search for patients matching a query string.

`patient_type`
string
Filter by patient type. Valid options are `all`, `dependent`, or `guardian`. If not provided, patients with no dependents will be returned.

GET
/api/clinic/search/patients

```
curl "https://api-us-snd.fullscript.io/api/clinic/
search/patients" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

200
OK

403
Forbidden

Address

The `address` object contains a record of all addresses that a patient has made. In this API you can look up the address on an individual patient.

List All Patient Addresses

The address object contains a record of all addresses that belong to a patient.

Arguments

`patient_id`
string
REQUIRED
Unique ID for the patient.

GET
/api/clinic/patients/{patient_id}/addresses

```
curl "https://api-us-snd.fullscript.io/api/clinic/
patients/{patient_id}/addresses" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

200
OK

403
Forbidden

404
Not Found

Orders

The `order` object contains a record of all completed or refunded orders that a patient has made.

List all orders

This resource returns all orders for the clinic and can be filtered by the following attributes.

Arguments

`patient_id`
string
Filter orders by `patient_id`.

`treatment_plan_id`
string
Filter orders by `treatment_plan_id`.

`completed_at`
string
Filter orders by `completed_at`. The date must be formatted as follows `yyyy-mm-dd`.

- less than: Return orders where the `completed_at` date is less than. `<yyyy-mm-dd`.
- greater than: Return orders where the `completed_at` date is greater than. `>yyyy-mm-dd`.
- within a range: Return orders where the `completed_at` date is with in a range. `yyyy-mm-dd..yyyy-mm-dd`.

`type`
string
Filter orders by `type`. The default value is `all`, which includes all types of orders. To filter specifically for lab orders, use the value `labs`.

`order_number`
string
Filter orders by `order_number`.

`sort_by`
string
Accepts one of the following arguments: `patient_id`, `treatment_plan_id`, `completed_at` and `order_number`.

`order_by`
string
Ordering defaults to `ASC` and can take an argument of `ASC` or `DESC`.

GET
/api/clinic/orders

```
curl "https://api-us-snd.fullscript.io/api/clinic/
orders" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

200
OK

403
Forbidden

Retrieve an Order

Retrieves an existing Order.

Arguments

`id`
string
REQUIRED
Unique identifier of the Order.

GET
/api/clinic/orders/{id}

```
curl "https://api-us-snd.fullscript.io/api/clinic/
orders/{id}" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

200
OK

403
Forbidden

404
Not Found

List all patient orders

The patient order object contains a record of all completed or refunded orders that belong to a patient.

Arguments

`patient_id`
string
REQUIRED
Unique ID for the Patient

GET
/api/clinic/patients/{patient_id}/orders

```
curl "https://api-us-snd.fullscript.io/api/clinic/
patients/{patient_id}/orders" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

200
OK

403
Forbidden

Treatment Plans

The `treatment plan` object is the way for practitioners create prescriptions in Fullscript. The API allows you to create treatment plans. It also allows you to list all treatment plans that belong to a patient and find individual treatment plans.

Activate a treatment plan

This endpoint is restricted
This endpoint is not available for general use. Access is highly restricted, and most requests will not be granted. If you believe your case is critical, you may submit a request for consideration. We will only respond if your request is relevant.
Activates a draft Treatment Plan for a patient.

Arguments

`treatment_plan_id`

string
REQUIRED
Unique ID for the Treatment plan.
PATCH
/api/clinic/treatment_plans/{treatment_plan_id}/activate

```
curl -X 'PATCH' "https://api-us-snd.fullscript.io/api/
clinic/treatment_plans/{treatment_plan_id}/activate" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

200
OK

403
Forbidden

404
Not Found

422
Unprocessable Entity

Cancel a treatment plan

Cancels an active Treatment Plan for a patient.

Arguments

`treatment_plan_id`
string
REQUIRED
Unique ID for the Treatment plan.
PATCH
/api/clinic/treatment_plans/{treatment_plan_id}/cancel

```
curl -X 'PATCH' "https://api-us-snd.fullscript.io/api/
clinic/treatment_plans/{treatment_plan_id}/cancel" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

200
OK

403
Forbidden

404
Not Found

422
Unprocessable Entity

List all patient treatment plans

This resource allows you to list all of a patient's treatment plans.

Arguments

`patient_id`
string
REQUIRED
Unique ID for the Patient

`sort_by`
string
Accepts one of the following arguments: `created_at` or `updated_at`.

`order_by`
string
Ordering defaults to `ASC` and can take an argument of `ASC` or `DESC`.

GET
/api/clinic/patients/{patient_id}/treatment_plans

```
curl "https://api-us-snd.fullscript.io/api/clinic/
patients/{patient_id}/treatment_plans" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

200
OK

403
Forbidden

404
Not Found

Create a patient treatment plan

This endpoint is restricted
This endpoint is not available for general use. Access is highly restricted, and most requests will not be granted. If you believe your case is critical, you may submit a request for consideration. We will only respond if your request is relevant.
Creates a new Treatment Plan for a patient.

Arguments

`patient_id`
string
REQUIRED
Unique ID for the Patient.

`practitioner_id`
string
Unique practitioner ID. Required if the current access token's resource owner type is `Staff` or `Clinic`. Otherwise defaults to the practitioner who owns the token, but can be specified to create the plan on behalf of a different practitioner.

`personal_message`
string
A personal message that a practitioner can attach to the treatment plan.

`state`
string
The state of the treatment plan. Takes an option of `draft` or `active`. Defaults to active if null. The value `draft` allows to create a draft treatment plan. The value `active` or null creates an active treatment plan.

`recommendations`
array
REQUIRED
Rx plan for a product.
Show child attributes

`metadata`
object
Metadata to be attached to the treatment_plan.
Show child attributes

POST
/api/clinic/patients/{patient_id}/treatment_plans

```
curl -X 'POST' "https://api-us-snd.fullscript.io/api/
clinic/patients/{patient_id}/treatment_plans" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX' \
  -d $'{
  "practitioner_id": "string",
  "personal_message": "string",
  "state": "string",
  "recommendations": [
    {
      "variant_id": "string",
      "units_to_purchase": "string",
      "refill": "string",
      "take_with": "string",
      "dosage": {
        "amount": "string",
        "frequency": "string",
        "duration": "string",
        "additional_info": "string",
        "format": "string",
        "time_of_day": "string"
      }
    }
  ],
  "metadata": {
    "id": "string"
  }
}'
```

Responses

201
Created

422
Unprocessable Entity

Retrieve a treatment plan

Retrieves an existing treatment plan. You need to supply the unique ID for the treatment plan.

Arguments

`id`
string
REQUIRED
Unique ID for the Treatment Plan

GET
/api/clinic/treatment_plans/{id}

```
curl "https://api-us-snd.fullscript.io/api/clinic/
treatment_plans/{id}" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

200
OK

403
Forbidden

404
Not Found

Update a treatment plan

Updates a Treatment Plan for a patient.

Arguments

`id`
string
REQUIRED
Unique ID for the Treatment plan.

`personal_message`
string
A personal message that a practitioner can attach to the treatment plan.

`recommendations`
array
Rx plan for a product.
Show child attributes

`resource_ids`
array
The Resource IDs to be attached to the Treatment Plan.

`metadata`
object
Metadata to be attached to the treatment_plan.
Show child attributes

PATCH

/api/clinic/treatment_plans/{id}

```
curl -X 'PATCH' "https://api-us-snd.fullscript.io/api/
clinic/treatment_plans/{id}" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX' \
  -d $'{
  "personal_message": "string",
  "recommendations": [
    {
      "variant_id": "string",
      "units_to_purchase": "string",
      "refill": "string",
      "take_with": "string",
      "dosage": {
        "amount": "string",
        "frequency": "string",
        "duration": "string",
        "additional_info": "string",
        "format": "string",
        "time_of_day": "string"
      }
    }
  ],
  "resource_ids": "array",
  "metadata": {
    "id": "string"
  }
}'
```

Responses

200
OK

403
Forbidden

422
Unprocessable Entity

Create an in-office checkout

The `in_office_checkout` object takes a `treatment_plan`. It uses the patient from the `treatment_plan` to:

a) clear out anything in the patient's cart.

b) populate the patient's cart with the treatment plan.

c) return a url on Fullscript so that a practitioner can fullfill an in-office checkout.

Arguments

`treatment_plan_id`
string
REQUIRED
Unique ID for the Treatment Plan

POST
/api/clinic/treatment_plans/{treatment_plan_id}/in_office_checkout

```
curl -X 'POST' "https://api-us-snd.fullscript.io/api/clinic/treatment_plans/{treatment_plan_id}/in_office_checkout" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

201
Created

404
Not Found

422
Unprocessable Entity

Dynamic link

Create a Draft Treatment Plan Link with Patient Matching UI

Returns a `redirect_url` that enables access to a draft treatment plan. This endpoint optionally finds or creates a patient record to associate with the treatment plan. It supports **dependent accounts** and includes a user interface that allows practitioners to manually match patients and practitioners if automatic matching is unsuccessful. When a match is automatically found, the patient is preselected for the practitioner.

OAuth scopes required:

- `clinic:write` - required for all requests to this endpoint.
- `patients:write` - required if the request may result in patient creation.

Arguments

`treatment_plan`
object
Treatment plan information for dynamic link.
Show child attributes

`patient`
object
Patient information used for creating a match with a Fullscript patient, or pre-filling information to create a new patient in Fullscript
Show child attributes

`entrypoint`
string
Specifies which page the redirect_url will open. Defaults to "catalog" if not specified.

- catalog: Starts the user on the product catalog page (default).
- labs: Starts the user on the labs page.

POST
/api/clinic/dynamic_links/gateway/treatment_plans

```
curl -X 'POST' "https://api-us-snd.fullscript.io/api/
clinic/dynamic_links/gateway/treatment_plans" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX' \
  -d $'{
  "treatment_plan": {
    "practitioner_id": "string"
  },
  "patient": {
    "id": "string",
    "external_id": "string",
    "email": "string",
    "first_name": "string",
    "last_name": "string",
    "date_of_birth": "string",
    "gender": "string",
    "mobile_number": "string",
    "discount": "string",
    "send_welcome_email": "string"
  },
  "entrypoint": "string"
}'
```

Responses

200
OK

400
Bad Request

Retrieve a new Treatment Plan link

Returns a `redirect_url` for a new treatment plan.

Arguments

No arguments...

GET
/api/clinic/dynamic_links/treatment_plans

```
curl "https://api-us-snd.fullscript.io/api/clinic/
dynamic_links/treatment_plans" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

200
OK

403

Forbidden

Create a Draft Treatment Plan Link

FYI

This endpoint does **not** support **dependent accounts**. If you are working with dependent accounts, use the Create a Draft Treatment Plan Link with Patient Matching UI endpoint instead.

Returns a `redirect_url` that enables access to a draft treatment plan. This endpoint optionally finds or creates a patient record to associate with the treatment plan.

OAuth scopes required:

- `clinic:write` - required for all requests to this endpoint.
- `patients:write` - required if the request may result in patient creation.

Arguments

`treatment_plan`
object
REQUIRED
Treatment plan information for dynamic link.
Show child attributes

`patient`
object
Find or create a Patient.
Show child attributes

`entrypoint`
string
Specifies which page the redirect_url will open. Defaults to "catalog" if not specified.

- catalog: (default) Starts the user on the product catalog page.
- labs: Starts the user on the labs page.

POST
/api/clinic/dynamic_links/treatment_plans

```
curl -X 'POST' "https://api-us-snd.fullscript.io/api/
clinic/dynamic_links/treatment_plans" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX' \
  -d $'{
  "treatment_plan": {
    "practitioner_id": "string"
  },
  "patient": {
    "email": "string",
    "first_name": "string",
    "last_name": "string",
    "date_of_birth": "string",
    "gender": "string",
    "mobile_number": "string",
    "discount": "string",
    "send_welcome_email": "string"
  },
  "entrypoint": "string"
}'
```

Responses

200
OK

400
Bad Request

422
Unprocessable Entity

Retrieve a Treatment Plan Link

Returns a `redirect_url` for editing an existing treatment plan. The treatment plan provided must have a `state` of `draft` or `active`.

Arguments

`id`
string
REQUIRED
Unique ID for the Treatment Plan

`entrypoint`
string
Target treatment plan page to open with the redirect_url. Defaults to "catalog"

- catalog: (default) Starts the user on the product catalog page.
- labs: Starts the user on the labs page.

GET
/api/clinic/dynamic_links/treatment_plans/{id}

```
curl "https://api-us-snd.fullscript.io/api/clinic/
dynamic_links/treatment_plans/{id}" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

200
OK

403
Forbidden

404
Not Found

Products

The `product` object contains details for all products on the Fullscript platform. The API allows you to list all products and find individual products.

List all products

This endpoint is restricted
This endpoint is not available for general use. Access is highly restricted, and most requests will not be granted. If you believe your case is critical, you may submit a request for consideration. We will only respond if your request is relevant.
This resource allows you to list all products.

Arguments

No arguments...
GET
/api/catalog/products

```
curl "https://api-us-snd.fullscript.io/api/catalog/
products" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

200
OK

403
Forbidden

Retrieve a product

Retrieves an existing product. If the practitioner has set custom dosage instructions for the product, the `custom_dosage` object is included in the response. Specify the pracitioner with `practitioner_id` (defaults to current practitioner, when applicable).

Arguments

`id`
string
REQUIRED
Unique identifier for the Product

`practitioner_id`
string
Unique Practitioner ID. Include to retrieve a specific practitioner's custom dosage instructions for the product. Defaults to the practitioner who owns current access token, if applicable (access token's resource owner type is `Practitioner`). Custom dosage is returned in the `custom_dosage` object.

GET
/api/catalog/products/{id}

```
curl "https://api-us-snd.fullscript.io/api/catalog/products/{id}" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

200
OK

403
Forbidden

404
Not Found

List similar products

This endpoint is restricted
This endpoint is not available for general use. Access is highly restricted, and most requests will not be granted. If you believe your case is critical, you may submit a request for consideration. We will only respond if your request is relevant.
This endpoint allows you to list similar products that have been curated by Fullscript's licensed health professionals. It is in order from most to least popular.

Arguments

`product_id`
string
REQUIRED
Unique identifier for the Product
GET
/api/catalog/products/{product_id}/similar

```
curl "https://api-us-snd.fullscript.io/api/catalog/
products/{product_id}/similar" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

200
OK

403
Forbidden

404
Not Found

Categories

The `category` object contains a record of all categories from a clinic.

List all category products

This resource allows you to list all of a category's products. The list is sorted in ascending order by the product's name, by default.

Arguments

`category_id`
string
REQUIRED
Unique ID for the Category.

`sort_by`
string
Accepts a `name` argument.

`order_by`
string
Ordering defaults to `ASC` and can take an argument of `ASC` or `DESC`.

GET
/api/clinic/categories/{category_id}/products

```
curl "https://api-us-snd.fullscript.io/api/clinic/
categories/{category_id}/products" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

200
OK

403
Forbidden

404
Not Found

Retrieve a Category

Retrieves an existing Category.

Arguments

`id`
string
REQUIRED
Unique identifier of the Category.
GET
/api/clinic/categories/{id}

```
curl "https://api-us-snd.fullscript.io/api/clinic/
categories/{id}" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

200
OK

403
Forbidden

404
Not Found

List all categories

This resource allows you to list all of a clinic's categories.

Arguments

No arguments...
GET
/api/clinic/categories

```
curl "https://api-us-snd.fullscript.io/api/clinic/
categories" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

200

OK

403
Forbidden

Product search

Search for products

This endpoint is restricted
This endpoint is not available for general use. Access is highly restricted, and most requests will not be granted. If you believe your case is critical, you may submit a request for consideration. We will only respond if your request is relevant.
This resource allows you to search for products using Fullscript's opensearch implementation. This endpoint takes into consideration brand segmentation (not all clinics have access to all brands) and will only show products and brands that your clinic has available to them.

Note: Filter arguments that accept an array can be specified in the query string like this:
`?allergen_ids[]=id_1&allergen_ids[]=id_2`

Arguments

`query`
string
Search for products matching a query string.

`allergen_ids`
string
Provide an array of allergen ids to filter results.

`brand_id`
string
Search for products by brand_id.

`ingredient_ids`
string
Provide an array of ingredient ids to filter results.

`supplement_type_ids`
string
Provide an array of supplement type ids to filter results.

`third_party_certification_ids`
string
Provide an array of third party certification ids to filter results.
GET
/api/catalog/search/products

```
curl "https://api-us-snd.fullscript.io/api/catalog/
search/products" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

200
OK

403
Forbidden

404
Not Found

Allergens

Attributes about a product as they relate specifically to ingredients known to cause allergic reactions / ingredients known to want to be avoided (e.g. 'Peanut free'). They are not attested by third parties.

The `allergen` object contains details for all allergens listed on the Fullscript platform. The API allows you to list all allergens and find individual allergens.

List all allergens

This resource allows you to list all allergens.

Arguments

No arguments...
GET
/api/catalog/allergens

```
curl "https://api-us-snd.fullscript.io/api/catalog/
allergens" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

200
OK

Retrieve an allergen

Retrieves details for an existing allergen.

Arguments

`id`
string
REQUIRED
Unique ID for the allergen

GET
/api/catalog/allergens/{id}

```
curl "https://api-us-snd.fullscript.io/api/catalog/
allergens/{id}" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

200
OK

403
Forbidden

404
Not Found

Brands

The `brand` object contains details for all brands on the Fullscript platform. The API allows you to list all brands and find individual brands.

List all brands

This resource allows you to list all brands.

Arguments

`available`
string
Takes an argument of `true` to return brands that have available products.

`clinic_permitted`
string
Takes an argument of `true` to return brands the current clinic has access to.

GET
/api/catalog/brands

```
curl "https://api-us-snd.fullscript.io/api/catalog/
brands" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

200
OK

403
Forbidden

Retrieve a brand

Retrieves details for an existing brand.

Arguments

`id`
string
REQUIRED
Unique ID for the Brand

GET
/api/catalog/brands/{id}

```
curl "https://api-us-snd.fullscript.io/api/catalog/
brands/{id}" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

200
OK

403
Forbidden

404
Not Found

Ingredients

Supplement ingredients that make up the product (e.g. 'Curcumin').

The `ingredient` object contains details for all ingredients listed on the Fullscript platform. The API allows you to list all ingredients and find individual ingredients.

List all Ingredients

This resource allows you to list all ingredients.

Arguments

No arguments...

GET
/api/catalog/ingredients

```
curl "https://api-us-snd.fullscript.io/api/catalog/
ingredients" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

200

OK

Retrieve an ingredient

Retrieves details for an existing ingredient.

Arguments

`id`
string
REQUIRED
Unique ID for the ingredient

GET
/api/catalog/ingredients/{id}

```
curl "https://api-us-snd.fullscript.io/api/catalog/
ingredients/{id}" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

200
OK

403
Forbidden

404
Not Found

Supplement Types

Groups of products based on similar attributes (e.g. 'Multivitamins').

The `supplement type` object contains details for all supplement types listed on the Fullscript platform. The API allows you to list all supplement types and find individual supplement types.

List all supplement types

This resource allows you to list all supplement types.

Arguments

No arguments...

GET

/api/catalog/supplement_types

```
curl "https://api-us-snd.fullscript.io/api/catalog/
supplement_types" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

200

OK

Retrieve a supplement type

Retrieves details for an existing supplement type.

Arguments

`id`

string

REQUIRED

Unique ID for the supplement type

GET

/api/catalog/supplement_types/{id}

```
curl "https://api-us-snd.fullscript.io/api/catalog/
supplement_types/{id}" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

200

OK

403

Forbidden

404
Not Found

Third Party Certifications

Third party certifications are product level filters that are assessed by a party external to the manufacturer (e.g. 'Certified Halal').

The `third party certification` object contains details for all third party certifications listed on the Fullscript platform. The API allows you to list all third party certifications and find individual third party certifications.

List all third party certifications

This resource allows you to list all third party certifications.

Arguments

No arguments...

GET
/api/catalog/third_party_certifications

```
curl "https://api-us-snd.fullscript.io/api/catalog/
third_party_certifications" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

200
OK

Retrieve a third party certification

Retrieves details for an existing third party certification.

Arguments

`id`
string

REQUIRED
Unique ID for the third party certification
GET
/api/catalog/third_party_certifications/{id}

```
curl "https://api-us-snd.fullscript.io/api/catalog/
third_party_certifications/{id}" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

200
OK

403
Forbidden

404
Not Found

Variants

The `variant` object contains details about product variants. The API allows you to find individual variants.

Retrieve a variant

Retrieves an existing variant.

Arguments

`id`
string
REQUIRED
Unique identifier for the Variant.
GET
/api/catalog/variants/{id}

```
curl "https://api-us-snd.fullscript.io/api/catalog/
variants/{id}" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

200
OK

403
Forbidden

404
Not Found

Webhooks

Webhooks allow your application to stay informed with what's happening on Fullscript. You can subscribe to events and be notified when they happen. Let's say one of your clinics updates a patient or creates a treatment plan; webhooks allow you to know about those events and take action if needed.

Using webhooks

You can register a single webhook URL through the API Dashboard and subscribe to multiple event types.

When the event occurs—say a treatment plan is created, a patient is updated, or a clinic key is revoked, etc.—Fullscript creates an `Event`. This `Event` contains all the relevant information about what just happened. It includes the event's type (ex: `treatment_plan.created`) and any relevant data to that event.

Once the event is created, Fullscript will send out a payload with the event data via an HTTP POST request to the webhook url that you defined in your account. Note that we only send events to a single endpoint per environment.

Setting up webhooks

Webhooks are configured through the API Dashboard. For each environment that you have setup, you can enter a URL to receive events. This should be a dedicated URL on your server that is setup to receive webhook notifications.

All urls need to use HTTPS. We will validate that your connection is secure and HIPAA compliant before sending any webhook data. For this to work, your server needs to be setup to support HTTPS with a valid certificate.

Responding to a webhook

To acknowledge that you received a webhook notification, your endpoint needs to respond with either a `200` or `201` HTTP status code. Any response codes outside of this range will tell us that the webhook has not been received and will be treated as a failure.

Your endpoint's body must also respond with your challenge token (available in the API Dashboard). This is a security measure to ensure that you own the url in question and it hasn't been hijacked. A successful response could look something like this:

```
HTTP 200 OK
Content-type: application/json
{ "challenge": "your-secret-challenge-token" }
```

We will attempt to deliver your webhooks up to 6 times with exponential back off. Webhooks cannot manually be retried, however you can make a query to the events endpoint to reconcile data in case of any missed events.

We will attempt to send event deliveries in order, however that is not a guarantee. We will also attempt to deliver webhooks within 30 minutes from when the event was created. In most cases however, this should be close to instantaneous.

The webhook deliveries endpoint has all the information you need to check how many times we've attempted to deliver an event, what the response received was, the status code, and other relevant information.

Time-outs

We set aggressive time-outs for webhook deliveries. If you have to do a bunch of complex logic with the event data, or need to make network requests, it's possible that the delivery attempt will time-out before we receive a `2xx` HTTP status code. To mitigate this, you may want to respond immediately with a `2xx` HTTP status code, and then perform your complex logic afterwards.

Handling multiple clinics and integrations

Each event payload comes with both a `clinic_id` and an `integration_id` (if relevant). The clinic key is the unique identifier for the clinic. Any clinic events (patient, practitioner, treatment plan, etc.) will be scoped to that clinic through the `clinic_id`.

Because it's possible for clinics to have multiple integrations enabled, the exact same events will be sent for each integration that a clinic has enabled with you. To mitigate unwanted duplication of events, the integration has to be activated before any webhook events will be sent. This means that it's been used as least once with a valid `X-FS-Clinic-Key` and the key has not been revoked.
The `integration_id` allows you to scope events to the relevant integration that is enabled.

Both the `clinic_id` and `integration_id` can be retrieved through the API at the clinic endpoint.

Multiple failures

In some circumstances we will disable your endpoint and stop trying to deliver events to it. This can happen when: we receive a `410` HTTP status code or if the delivery attempts are consistently failing for more than 24 hours. In that case your webhook endpoint may be disabled and we will no longer send events to that url.

Re-enabling a disabled webhook can be done through the API Dashboard once you have resolved the issues.

You can see the status (both failed and successful delivery attempts) through the webhook deliveries endpoint.

Webhook versioning

Webhook events follow the same logic as our versioning scheme in the API and it respects the `patch_version` that you're on.

Webhooks Security

Checking the `Fullscript-Signature` header

Fullscript sends a header in the webhook request that can be used in conjunction with your `Webhook Secret Key` to verify the payload. Here is an example of the header:

```
Fullscript-Signature:
```

```
t=1591826856,v1=0c262932b0ac6b4952e2fe24fdf419313984a66
f6f442e0b8ec4cb87f2a107ad
```

This represents the format: `t=<timestamp>,v1=<signature>`

The `signature` is generated using a hash-based message authentication code (HMAC) with SHA-256. This hash is generated using 3 things:

- A UTC `timestamp`
- The `request_body` (The POST message's JSON payload string)
- Your `Webhook Secret Key`

The `timestamp` and `request_body` are combined to create the payload provided to the hash function. The `Webhook Secret Key` is used as the key. The same hash can be computed and compared to the `signature` to verify the request.

To recompute and compare the hash, follow these steps:

- Extract the `timestamp` and `signature` from the header.
- Create the `payload` by combining the `timestamp` and `request_body` with a single `.`

Example: `payload = '1591826856.{"event":{...<more_json>}}'`

- Compute an HMAC with the SHA256 hash function. Use the `Webhook Secret Key` as the key. Use the `payload` string as the message.
- Compare the result with the `signature`.
- If they match, the request payload is legitimate.

Note: The timestamp is generated directly before we send the payload. It is included in the hash payload in order to help prevent replay attacks. We recommend using this to verify the age of a request with a reasonable tolerance. 5 minutes is usually a good default.

List All Webhook Deliveries

This endpoint lists all `webhook_deliveries` from the last 30 days.

Arguments

`attempted_at`
string
Filter by `attempted_at` date. The date must be formatted as follows `yyyy-mm-dd`.

- less than: Return webhook deliveries where the `attempted_at` date is less than. `<yyyy-mm-dd`.
- greater than: Return webhook deliveries where the `attempted_at` date is greater than. `>yyyy-mm-dd`.
- within a range: Return webhook deliveries where the `attempted_at` date is within a range. `yyyy-mm-dd..yyyy-mm-dd`.

`event_id`
string
Filter webhook deliveries by `event_id`.

`successfully_delivered`
string
Filter webhook deliveries by whether they have been successfully delivered or not. Accepts a string of `true` or `false`

`sort_by`
string
Accepts one of the following arguments: `event_id` `attempted_at`.

`order_by`
string
Ordering defaults to `ASC` and can take an argument of `ASC` or `DESC`.

GET
/api/webhooks/deliveries

```
curl "https://api-us-snd.fullscript.io/api/webhooks/deliveries" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

`200`
OK

`403`
Forbidden

Events

Events allow your integration to stay informed with what's happening on Fullscript. Let's say one of your clinics updates a patient, or creates a treatment plan. In each case an `Event` will be created which allows you to take action if need be.

This `Event` has all of the relevant information about what just happened. It includes the event's type (say `treatment_plan.created`) and any relevant data to that event.

Note that we only keep a record of events for the past 30 days.

Also know that events are never created for actions made through the API. We assume that, since you initiated the event, you already know about it! 😀

List All ClinicKey Events

This endpoint lists all `clinic_key` events from the last 30 days.

- A `clinic_key.revoked` event happens when a clinic revokes access to your integration.
- A `clinic_key.activated` event happens when the first request is made through the API with a valid `X-FS-Clinic-Key`—activating the integration.

Arguments

`created_at`
string
Filter by `created_at` date. The date must be formatted as follows `yyyy-mm-dd`.

- less than: Return events where the `created_at` date is less than. `<yyyy-mm-dd`.

- greater than: Return events where the `created_at` date is greater than. `>yyyy-mm-dd`.
- within a range: Return events where the `created_at` date is within a range. `yyyy-mm-dd..yyyy-mm-dd`.

`event_type`
string
Filter events by `event_type`.

`sort_by`
string
Accepts one of the following arguments: `event_type` `created_at`.

`order_by`
string
Ordering defaults to `ASC` and can take an argument of `ASC` or `DESC`.

GET
/api/events/clinic_keys

```
curl "https://api-us-snd.fullscript.io/api/events/
clinic_keys" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

200
OK

403
Forbidden

Retrieve an Event

This endpoint retrieves details from an Event. Note that event's can only be retrieved if they were made within the last 30 days.

Arguments

`id`
string
REQUIRED
Unique identifier of the Event

GET

/api/events/{id}

```
curl "https://api-us-snd.fullscript.io/api/events/{id}" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

200
OK

403
Forbidden

404
Not Found

List All Lab Order updated Events

This endpoint lists all `lab order` events from the last 30 days.

- A `lab_order.updated` event happens when a lab test result is available or has been ammended.

Arguments

`created_at`
string
Filter by `created_at` date. The date must be formatted as follows `yyyy-mm-dd`.

- less than: Return events where the `created_at` date is less than. `<yyyy-mm-dd`.
- greater than: Return events where the `created_at` date is greater than. `>yyyy-mm-dd`.
- within a range: Return events where the `created_at` date is within a range. `yyyy-mm-dd..yyyy-mm-dd`.

`sort_by`

string
Accepts a `created_at`. argument.

`order_by`
string
Ordering defaults to `ASC` and can take an argument of `ASC` or `DESC`.

GET
/api/events/lab_orders

```
curl "https://api-us-snd.fullscript.io/api/events/
lab_orders" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

200
OK

403
Forbidden

List All Order Events

This endpoint lists all `order` events from the last 30 days.

- An `order.placed` event happens when an order has been placed.

Arguments

`created_at`
string
Filter by `created_at` date. The date must be formatted as follows `yyyy-mm-dd`.

- less than: Return events where the `created_at` date is less than. `<yyyy-mm-dd`.
- greater than: Return events where the `created_at` date is greater than. `>yyyy-mm-dd`.
- within a range: Return events where the `created_at` date is

within a range. `yyyy-mm-dd..yyyy-mm-dd`.

`event_type`
string
Filter events by `event_type`.

`sort_by`
string
Accepts one of the following arguments: `event_type` `created_at`.

`order_by`
string
Ordering defaults to `ASC` and can take an argument of `ASC` or `DESC`.

GET
/api/events/orders

```
curl "https://api-us-snd.fullscript.io/api/events/
orders" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

200
OK

403
Forbidden

List All Patient Events

This endpoint lists all `patient` events from the last 30 days.

- A `patient.created` event happens when a patient is created.
- A `patient.updated` event happens when a patient is updated.
- A `patient.emancipated` event happens when a patient completes emancipation.

Arguments

`created_at`
string
Filter by `created_at` date. The date must be formatted as follows `yyyy-mm-dd`.

- less than: Return events where the `created_at` date is less than. `<yyyy-mm-dd`.

- greater than: Return events where the `created_at` date is greater than. `>yyyy-mm-dd`.
- within a range: Return events where the `created_at` date is within a range. `yyyy-mm-dd..yyyy-mm-dd`.

`event_type`
string
Filter events by `event_type`.

`sort_by`
string
Accepts one of the following arguments: `event_type` `created_at`.

`order_by`
string
Ordering defaults to `ASC` and can take an argument of `ASC` or `DESC`.

GET
/api/events/patients

```
curl "https://api-us-snd.fullscript.io/api/events/
patients" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

200
OK

403
Forbidden

List All Product Events

This endpoint lists all `product` events from the last 30 days.

- A `product.created` event happens when a product is created.
- A `product.updated` event happens when a product or one of its variants is updated.
- A `product.description.updated` event happens when a product's description is updated.

Arguments

`created_at`
string
Filter by `created_at` date. The date must be formatted as follows `yyyy-mm-dd`.

- less than: Return events where the `created_at` date is less than. `<yyyy-mm-dd`.
- greater than: Return events where the `created_at` date is greater than. `>yyyy-mm-dd`.
- within a range: Return events where the `created_at` date is within a range. `yyyy-mm-dd..yyyy-mm-dd`.

`event_type`
string
Filter events by `event_type`.

`sort_by`
string
Accepts one of the following arguments: `event_type` `created_at`.

`order_by`
string
Ordering defaults to `ASC` and can take an argument of `ASC` or `DESC`.

GET
/api/events/products

```
curl "https://api-us-snd.fullscript.io/api/events/
products" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

200
OK

403
Forbidden

List All TreatmentPlan Events

This endpoint lists all `treatment_plan` events from the last 30 days.

- A `treatment_plan.created` event happens when a Treatment Plan is created.
- A `treatment_plan.updated` event happens when an active Treatment Plan is modified, such as when it is cancelled.
  Note: This webhook does not trigger when a draft becomes active, and patient or practitioner changes are not supported on active plans.

- A `treatment_plan.recommendation.updated` event happens when a Treatment Plan's recommendation is updated.

Arguments

`created_at`
string
Filter by `created_at` date. The date must be formatted as follows `yyyy-mm-dd`.

- less than: Return events where the `created_at` date is less than. `<yyyy-mm-dd`.
- greater than: Return events where the `created_at` date is greater than. `>yyyy-mm-dd`.
- within a range: Return events where the `created_at` date is within a range. `yyyy-mm-dd..yyyy-mm-dd`.

`event_type`
string
Filter events by `event_type`.

`sort_by`
string
Accepts one of the following arguments: `event_type` `created_at`.

`order_by`
string
Ordering defaults to `ASC` and can take an argument of `ASC` or `DESC`.

GET
/api/events/treatment_plans

```
curl "https://api-us-snd.fullscript.io/api/events/treatment_plans" \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer XXXXXXXXXXXXXXXXXXXX'
```

Responses

200

OK

403

Forbidden