This version of GitHub Enterprise was discontinued on 2023-03-15. No patch releases will be made, even for critical security issues. For better performance, improved security, and new features, upgrade to the latest version of GitHub Enterprise. For help with the upgrade, contact GitHub Enterprise support.

Building a "Login with GitHub" button with a GitHub App

In this article

Follow this tutorial to write Ruby code to generate a user access token via the web application flow for your GitHub App.

Introduction

This tutorial demonstrates how to build a "Login with GitHub" button for a website. The website will use a GitHub App to generate a user access token via the web application flow. Then, the website uses the user access token to make API requests on behalf of the authenticated user.

This tutorial uses Ruby, but you can use the web application flow with any programming language that is used for web development.

About web application flow and user access tokens

Your app should use a user access token if you want to attribute the app's actions to a user. For more information, see "Authenticating with a GitHub App on behalf of a user."

There are two ways to generate a user access token for a GitHub App: web application flow and device flow. If your app has access to a web interface, you should use web application flow. If your app does not have access to a web interface, you should use device flow instead. For more information, see "Generating a user access token for a GitHub App" and "Building a CLI with a GitHub App."

Prerequisites

This tutorial assumes that you have already registered a GitHub App. For more information about registering a GitHub App, see "Registering a GitHub App."

Before following this tutorial, you must set a callback URL for your app. This tutorial uses

a local Sinatra server with the default URL of `http://localhost:4567`. For example, to work with the default URL for a local Sinatra application, your callback URL can be `http://localhost:4567/github/callback`. Once you are ready to deploy your app, you can change the callback URL to use your live server address. For more information about updating the callback URL for your app, see "Modifying a GitHub App registration" and "About the user authorization callback URL."

This tutorial assumes that you have a basic understanding of Ruby and of the Ruby template system, ERB. For more information, see Ruby and ERB.

Install dependencies

This tutorial uses the Ruby gem, Sinatra, to create a web application with Ruby. For more information, see the Sinatra README.

This tutorial uses the Ruby gem, dotenv, to access values stored in a `.env` file. For more information, see the dotenv README.

To follow this tutorial, you must install the Sinatra and dotenv gems in your Ruby project. For example, you can do this with Bundler:

1. If you don't already have Bundler installed, run the following command in your terminal:

   ```
   gem install bundler
   ```

2. If you don't already have a Gemfile for your app, run the following command in your terminal:

   ```
   bundle init
   ```

3. If you don't already have a Gemfile.lock for your app, run the following command in your terminal:

   ```
   bundle install
   ```

4. Install the gems by running the following commands in your terminal:

   ```
   bundle add sinatra
   ```

   ```
   bundle add dotenv
   ```

Store the client ID and client secret

This tutorial will show you how to store the client ID and client secret in environment variables and access them with `ENV.fetch`. When you deploy your app, you will want to change how you store the client ID and client secret. For more information, see "Securely store your client secret."

1. Navigate to your account settings.
   - For a GitHub App owned by a personal account, in the upper-right corner of any

page, click your profile photo, then click **Settings**.
- For a GitHub App owned by an organization, in the upper-right corner of any page, click your profile photo, then click **Your organizations**. Then, to the right of the organization, click **Settings**.

2. In the left sidebar, click **Developer settings**.
3. In the left sidebar, click **GitHub Apps**.
4. Next to the GitHub App that you want to work with, click **Edit**.
5. On the app's settings page, find the client ID for your app. You will add it to a `.env` file in a following step. Note that the client ID is different from the app ID.
6. On the app's settings page, click **Generate a new client secret**. You will add the client secret to a `.env` file in a following step.
7. Create a file called `.env` at the same level as your `Gemfile`.
8. If your project doesn't already have a `.gitignore` file, create a `.gitignore` file at the same level as your `Gemfile`.
9. Add `.env` to your `.gitignore` file. This will prevent you from accidentally committing your client secret. For more information about `.gitignore` files, see "Ignoring files."
10. Add the following contents to your `.env` file. Replace `YOUR_CLIENT_ID` with the client ID of your app. Replace `YOUR_CLIENT_SECRET` with the client secret for your app.

```
CLIENT_ID="YOUR_CLIENT_ID"
CLIENT_SECRET="YOUR_CLIENT_SECRET"
```

Add code to generate a user access token

To get a user access token, you first need to prompt the user to authorize your app. When a user authorizes your app, they are redirected to the callback URL for your app. The request to the callback URL includes a `code` query parameter. When your app gets a request to serve that callback URL, you can exchange the `code` parameter for a user access token.

These steps lead you through writing code to generate a user access token. To skip ahead to the final code, see "Full code example."

1. In the same directory as your `.env` file, create a Ruby file to hold the code that will generate a user access token. This tutorial will name the file `app.rb`.
2. At the top of `app.rb`, add these dependencies:

Ruby

```ruby
require "sinatra"
require "dotenv/load"
require "net/http"
require "json"
```

The `sinatra` and `dotenv/load` dependencies use the gems that you installed

earlier. `net/http` and `json` are part of the Ruby standard library.

3. Add the following code to `app.rb`, to get your app's client ID and client secret from your `.env` file.

```ruby
CLIENT_ID = ENV.fetch("CLIENT_ID")
CLIENT_SECRET = ENV.fetch("CLIENT_SECRET")
```

4. Add the following code to `app.rb` to display a link that will prompt users to authenticate your app.

```ruby
get "/" do
  link = '<a href="http(s)://HOSTNAME/login/oauth/authorize?client_id=<%= CLIEN
  erb link
end
```

5. Add the following code to `app.rb` to handle requests to your app's callback URL and get the `code` parameter from the request. Replace `CALLBACK_URL` with the callback URL for your app, minus the domain. For example, if your callback URL is `http://localhost:4567/github/callback`, replace `CALLBACK_URL` with `/github/callback`.

```ruby
get "CALLBACK_URL" do
  code = params["code"]
  render = "Successfully authorized! Got code #{code}."
  erb render
end
```

Currently, the code just renders a message along with the `code` parameter. The following steps will expand this code block.

6. Optionally, check your progress:

`app.rb` now looks like this, where `CALLBACK_URL` is the callback URL for your app, minus the domain:

```ruby
require "sinatra"
require "dotenv/load"
require "net/http"
require "json"

CLIENT_ID = ENV.fetch("CLIENT_ID")
CLIENT_SECRET = ENV.fetch("CLIENT_SECRET")

get "/" do
  link = '<a href="http(s)://HOSTNAME/login/oauth/authorize?client_id=<%= CLIEN
  erb link
end

get "CALLBACK_URL" do
```

```
  code = params["code"]
  render = "Successfully authorized! Got code #{code}."
  erb render
end
```

a. In your terminal, from the directory where `app.rb` is stored, run `ruby app.rb`. A local Sinatra server should start.

b. In your browser, navigate to `http://localhost:4567`. You should see a link with the text "Login with GitHub".

c. Click on the "Login with GitHub" link.

   If you have not authorized the app, clicking on the link should take you to `http(s)://HOSTNAME/login/oauth/authorize?client_id=CLIENT_ID`, where `CLIENT_ID` is the client ID of your app. This is a GitHub page that prompts users to authorize your app. If you click the button to authorize your app, you will go to the callback URL for your app.

   If you previously authorized your app and the authorization has not been revoked, you will skip the authorization prompt and go directly to the callback URL instead. You can revoke your previous authorization if you want to see the authorization prompt. For more information, see "Reviewing and revoking authorization of GitHub Apps."

d. The callback URL page, reached by clicking the "Login with GitHub" link and then authorizing the app if prompted to do so, should display the text similar to "Successfully authorized! Got code agc622abb6135be5d1f2."

e. In your terminal where Sinatra is running, stop the server by entering `Ctrl` + `C`.

7. Replace the content of `app.rb` with the following code, where `CALLBACK_URL` is the callback URL for your app, minus the domain.

   This code adds logic to exchange the `code` parameter for a user access token:

   - The `parse_response` function parses the response from the GitHub API.
   - The `exchange_code` function exchanges the `code` parameter for a user access token.
   - The handler for the callback URL request now calls `exchange_code` to exchange the code parameter for a user access token.
   - The callback page now shows text to indicate that a token was generated. If the token generation was not successful, the page will indicate that failure.

Ruby

```ruby
require "sinatra"
require "dotenv/load"
require "net/http"
require "json"

CLIENT_ID = ENV.fetch("CLIENT_ID")
CLIENT_SECRET = ENV.fetch("CLIENT_SECRET")

def parse_response(response)
  case response
  when Net::HTTPOK
    JSON.parse(response.body)
  else
    puts response
```

```
    puts response.body
    {}
  end
end

def exchange_code(code)
  params = {
    "client_id" => CLIENT_ID,
    "client_secret" => CLIENT_SECRET,
    "code" => code
  }
  result = Net::HTTP.post(
    URI("http(s)://HOSTNAME/login/oauth/access_token"),
    URI.encode_www_form(params),
    {"Accept" => "application/json"}
  )

  parse_response(result)
end

get "/" do
  link = '<a href="http(s)://HOSTNAME/login/oauth/authorize?client_id=<%= CLIEN
  erb link
end

get "CALLBACK_URL" do
  code = params["code"]

  token_data = exchange_code(code)

  if token_data.key?("access_token")
    token = token_data["access_token"]

    render = "Successfully authorized! Got code #{code} and exchanged it for a
    erb render
  else
    render = "Authorized, but unable to exchange code #{code} for token."
    erb render
  end
end
```

8. Optionally, check your progress:

   a. In your terminal, from the directory where `app.rb` is stored, run `ruby app.rb`. A local Sinatra server should start.

   b. In your browser, navigate to `http://localhost:4567`. You should see a link with the text "Login with GitHub".

   c. Click on the "Login with GitHub" link.

   d. If prompted to do so, authorize your app.

   e. The callback URL page, reached by clicking the "Login with GitHub" link and then authorizing the app if prompted to do so, should display the text similar to "Successfully authorized! Got code 4acd44861aeda86dacce and exchanged it for a user access token ending in 2zU5kQziE."

   f. In your terminal where Sinatra is running, stop the server by entering `Ctrl` + `C`.

9. Now that you have a user access token, you can use the token to make API requests on behalf of the user. For example:

   Add this function to `app.rb` to get information about the user with the `/user` REST

API endpoint:

Ruby

```ruby
def user_info(token)
  uri = URI("http(s)://HOSTNAME/api/v3/user")

  result = Net::HTTP.start(uri.host, uri.port, use_ssl: true) do |http|
    body = {"access_token" => token}.to_json

    auth = "Bearer #{token}"
    headers = {"Accept" => "application/json", "Content-Type" => "application/j

    http.send_request("GET", uri.path, body, headers)
  end

  parse_response(result)
end
```

Update the callback handler to call the `user_info` function and to display the user's name and GitHub login. Remember to replace `CALLBACK_URL` with the callback URL for your app, minus the domain.

Ruby

```ruby
get "CALLBACK_URL" do
  code = params["code"]

  token_data = exchange_code(code)

  if token_data.key?("access_token")
    token = token_data["access_token"]

    user_info = user_info(token)
    handle = user_info["login"]
    name = user_info["name"]

    render = "Successfully authorized! Welcome, #{name} (#{handle})."
    erb render
  else
    render = "Authorized, but unable to exchange code #{code} for token."
    erb render
  end
end
```

10. Check your code against the full code example in the next section. You can test your code by following the steps outlined in the "Testing" section below the full code example.

Full code example

This is the full code example that was outlined in the previous section.

Replace `CALLBACK_URL` with the callback URL for your app, minus the domain. For example, if your callback URL is `http://localhost:4567/github/callback`, replace `CALLBACK_URL` with `/github/callback`.

Ruby

```ruby
require "sinatra"
require "dotenv/load"
require "net/http"
require "json"

CLIENT_ID = ENV.fetch("CLIENT_ID")
CLIENT_SECRET = ENV.fetch("CLIENT_SECRET")

def parse_response(response)
  case response
  when Net::HTTPOK
    JSON.parse(response.body)
  else
    puts response
    puts response.body
    {}
  end
end

def exchange_code(code)
  params = {
    "client_id" => CLIENT_ID,
    "client_secret" => CLIENT_SECRET,
    "code" => code
  }
  result = Net::HTTP.post(
    URI("http(s)://HOSTNAME/login/oauth/access_token"),
    URI.encode_www_form(params),
    {"Accept" => "application/json"}
  )

  parse_response(result)
end

def user_info(token)
  uri = URI("http(s)://HOSTNAME/api/v3/user")

  result = Net::HTTP.start(uri.host, uri.port, use_ssl: true) do |http|
    body = {"access_token" => token}.to_json

    auth = "Bearer #{token}"
    headers = {"Accept" => "application/json", "Content-Type" => "application/json"

    http.send_request("GET", uri.path, body, headers)
  end

  parse_response(result)
end

get "/" do
  link = '<a href="http(s)://HOSTNAME/login/oauth/authorize?client_id=<%= CLIENT_ID
  erb link
end

get "CALLBACK_URL" do
  code = params["code"]

  token_data = exchange_code(code)

  if token_data.key?("access_token")
    token = token_data["access_token"]

    user_info = user_info(token)
    handle = user_info["login"]
    name = user_info["name"]

    render = "Successfully authorized! Welcome, #{name} (#{handle})."
    erb render
  else
    render = "Authorized, but unable to exchange code #{code} for token."
```

```
    erb render
  end
end
```

Testing

This tutorial assumes that your app code is stored in a file named `app.rb` and that you are using the default URL for a local Sinatra application, `http://localhost:4567`.

1. In your terminal, from the directory where `app.rb` is stored, run `ruby app.rb`. A local Sinatra server should start.
2. In your browser, navigate to `http://localhost:4567`. You should see a link with the text "Login with GitHub".
3. Click on the "Login with GitHub" link.

   If you have not authorized the app, clicking on the link should take you to `http(s)://HOSTNAME/login/oauth/authorize?client_id=CLIENT_ID`, where `CLIENT_ID` is the client ID of your app. This is a GitHub page that prompts users to authorize your app. If you click the button to authorize your app, you will go to the callback URL for your app.

   If you previously authorized your app and the authorization has not been revoked, you will skip the authorization prompt and go directly to the callback URL instead. You can revoke your previous authorization if you want to see the authorization prompt. For more information, see "Reviewing and revoking authorization of GitHub Apps."
4. The callback URL page, reached by clicking the "Login with GitHub" link and then authorizing the app if prompted to do so, should display the text similar to "Successfully authorized! Welcome, Mona Lisa (octocat)."
5. In your terminal where Sinatra is running, stop the server by entering `Ctrl` + `C`.

Next steps

Securely store your client secret

You should never publicize your app's client secret. This tutorial stored the client secret in a gitignored `.env` file and accessed the value with `ENV.fetch`. When you deploy your app, you should choose a secure way to store the client secret and update your code to get the value accordingly.

For example, you can store the secret in an environment variable on the server where your application is deployed. You can also use a secret management service like Azure Key Vault.

Update the callback URL for deployment

This tutorial used a callback URL starting with `http://localhost:4567`. However, `http://localhost:4567` is only available locally to your computer when you start the Sinatra server. Before you deploy your app, you should update the callback URL to use the callback URL that you use in production. For more information about updating the callback URL for your app, see "Modifying a GitHub App registration" and "About the user authorization callback URL."

Handle multiple callback URLs

This tutorial used a single callback URL, but your app can have up to 10 callback URLs. If you want to use multiple callback URLs:

- Add the additional callback URLs to your app. For more information about adding callback URLs, see "Modifying a GitHub App registration."
- When you link to `http(s)://HOSTNAME/login/oauth/authorize`, use the `redirect_uri` query parameter to redirect users to the desired callback URL. For more information, see "Generating a user access token for a GitHub App."
- In your app code, handle each callback URL, similar to the code block starting in `get "CALLBACK_URL" do`.

Specify additional parameters

When you link to `http(s)://HOSTNAME/login/oauth/authorize`, you can pass additional query parameters. For more information, see "Generating a user access token for a GitHub App."

Unlike a traditional OAuth token, the user access token does not use scopes so you cannot specify scopes via the `scope` parameter. Instead, it uses fine-grained permissions. A user access token only has permissions that both the user and the app have.

Adjust the code to meet your app's needs

This tutorial demonstrated how to display information about the authenticated user, but you can adjust this code to take other actions. Remember to update your app's permissions if your app needs additional permissions for the API requests that you want to make. For more information, see "Choosing permissions for a GitHub App."

This tutorial stored all of the code into a single file, but you may want to move functions and components into separate files.

Securely store tokens

This tutorial generates a user access token. Unless you opted out of expiration for user access tokens, the user access token will expire after eight hours. You will also receive a refresh token that can regenerate a user access token. For more information, see "Refreshing user access tokens."

If you plan on interacting further with GitHub's APIs, you should store the token for future use. If you choose to store the user access token or refresh token, you must store it securely. You should never publicize the token.

For more information, see "Best practices for creating a GitHub App."

Follow best practices

You should aim to follow best practices with your GitHub App. For more information, see "Best practices for creating a GitHub App."

Legal

© 2023 GitHub, Inc. Terms Privacy Status Pricing Expert services Blog