How to implement Rails API authentication with Devise and Doorkeeper

6 December 2020 by **Axel Kee**

Most of the time when we implement API endpoints on our Rails app, we want to limit access of these API to authorized users only, there's a few strategy for authenticating user through API, ranging from a simple token authentication to a fullblown OAuth provider with JWT.

In this tutorial, we will implement an OAuth provider for API authentication on the same Rails app we serve the user, using Devise and Doorkeeper gem.

After this tutorial, you would be able to implement Devise sign in/sign up on Rails frontend, and Doorkeeper OAuth (login, register) on the API side for mobile app client, or a separate frontend client like React etc.

This tutorial assume that you have some experience using Devise and your Rails app will both have a frontend UI and API for users to register and sign in. We can also use Doorkeeper to allow third party to create their own OAuth application on our own Rails app platform, but that is out of the scope of this article, as this article will focus on creating our own OAuth application for self consumption only.

Table of contents

Scaffold a model

Let's start with some scaffolding so we can have a model, controller and view for CRUD, you can skip this section if you already have an existing Rails app.

```
rails g scaffold bookmarks title:string url:string
```

then in **routes.rb** , set the root path to 'bookmarks#index'. Devise requires us to set a root path in routes to work.

```
# config/routes.rb

root 'bookmarks#index'
```

Now we have a sample CRUD Rails app, we can move on to the next step.

Setup Devise gem

Add devise gem in the **Gemfile** :

```
# Gemfile
# ...
gem 'devise', '~> 4.7.3'
```

and run `bundle install` to install it.

Next, run the Devise installation generator :

```
rails g devise:install
```

Then we create the user model (or any other model name you are using like admin, staff etc) using Devise :

```
rails g devise User
```

You can customize the devise features you want in the generated migration file, and also in the User model file.

Then run `rake db:migrate` to create the users table.

Now we have the Devise user set up, we can add **authenticate_user!** to **bookmarks_controller.rb** so only logged in users can view the controller now.

```
# app/controllers/bookmarks_controller.rb

class BookmarksController < ApplicationController
  before_action :authenticate_user!

  # ....
end
```

Next we will move to the main part, which is setting up authentication for the API using Doorkeeper gem.

Setup Doorkeeper gem

Add doorkeeper gem in the **Gemfile** :

```
# Gemfile
# ...
gem 'doorkeeper', '~> 5.4.0'
```

and run `bundle install` to install it.

Next, run the Doorkeeper installation generator :

```
rails g doorkeeper:install
```

This will generate the configuration file for Doorkeeper in **config/initializers/doorkeeper.rb**, which we will customize later.

Next, run the Doorkeeper migration generator :

```
rails g doorkeeper:migration
```

This will generate a migration file for Doorkeeper in **db/migrate/..._create_doorkeeper_tables.rb** .

We will customize the migration file as we won't need all the tables / attributes generated.

Open the **...._create_doorkeeper_tables.rb** migration file, then edit to make it look like below :

```
# frozen_string_literal: true

class CreateDoorkeeperTables < ActiveRecord::Migration[6.0]
  def change
    create_table :oauth_applications do |t|
      t.string  :name,    null: false
      t.string  :uid,     null: false
      t.string  :secret,  null: false

      # Remove `null: false` if you are planning to use grant flows
      # that doesn't require redirect URI to be used during authorization
      # like Client Credentials flow or Resource Owner Password.
      t.text    :redirect_uri
      t.string  :scopes,       null: false, default: ''
      t.boolean :confidential, null: false, default: true
      t.timestamps             null: false
    end

    add_index :oauth_applications, :uid, unique: true

    create_table :oauth_access_tokens do |t|
```

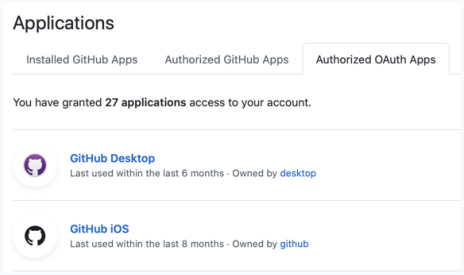
```
      t.references :resource_owner, index: true

      # Remove `null: false` if you are planning to use Password
      # Credentials Grant flow that doesn't require an application.
      t.references :application,    null: false

      t.string :token, null: false

      t.string   :refresh_token
      t.integer  :expires_in
      t.datetime :revoked_at
      t.datetime :created_at, null: false
      t.string   :scopes

      # The authorization server MAY issue a new refresh token, in which cas
      # *the client MUST discard the old refresh token* and replace it with
      # new refresh token. The authorization server MAY revoke the old
      # refresh token after issuing a new refresh token to the client.
      # @see https://tools.ietf.org/html/rfc6749#section-6
      #
      # Doorkeeper implementation: if there is a `previous_refresh_token` co
      # refresh tokens will be revoked after a related access token is used.
      # If there is no `previous_refresh_token` column, previous tokens are
      # revoked as soon as a new access token is created.
      #
      # Comment out this line if you want refresh tokens to be instantly
      # revoked after use.
      t.string   :previous_refresh_token, null: false, default: ""
    end

    add_index :oauth_access_tokens, :token, unique: true
    add_index :oauth_access_tokens, :refresh_token, unique: true
    add_foreign_key(
      :oauth_access_tokens,
      :oauth_applications,
      column: :application_id
    )
  end
end
```

The modification I did on the migration file :

1. Remove **null: false** on the **redirect_uri** for oauth_applications table. As we are using the OAuth for API authentication, we won't need to redirect the user to a callback page (like after you sign in with Google / Apple / Facebook on an app, they will redirect to a page usually).
2. Remove the creation of table **oauth_access_grants** , along with its related index and foreign key.

The OAuth application table is used to keep track of the application we created to use for authentication. For example, we can create three application, one for Android app client, one for iOS app client and one for React frontend, this way we can know which clients the users are using. If you only need one client (eg: web frontend), it is fine too.

Here's an example of Github OAuth applications :



Next, run `rake db:migrate` to add these tables into database.

Next, we will customize the Doorkeeper configuration.

Customize Doorkeeper configuration

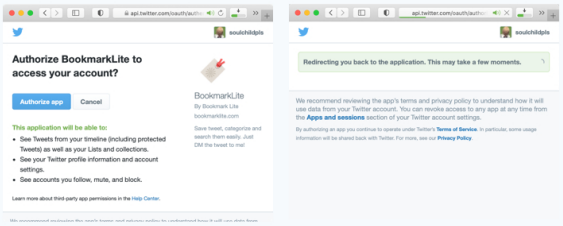
Open **config/initializers/doorkeeper.rb** , and edit the following.

Comment out or remove the block for **resource_owner_authenticator** at the top of the file.

```
#config/initializers/doorkeeper.rb

Doorkeeper.configure do
  # Change the ORM that doorkeeper will use (requires ORM extensions install
  # Check the list of supported ORMs here: https://github.com/doorkeeper-gem
  orm :active_record

  # This block will be called to check whether the resource owner is authent
```

```
  # resource_owner_authenticator do
    # raise "Please configure doorkeeper resource_owner_authenticator block
    # Put your resource owner authentication logic here.
    # Example implementation:
    #   User.find_by(id: session[:user_id]) || redirect_to(new_user_session_
  # end
```

The **resouce_owner_authenticator** block is used to get the authenticated user information or redirect the user to login page from OAuth, for example like this Twitter OAuth page :



As we are going to exchange OAuth token by using user login credentials (email + password) on the API, we don't need to implement this block, so we can comment it out.

To tell Doorkeeper we are using user credentials to login, we need to implement the **resource_owner_from_credentials** block like this :

```
#config/initializers/doorkeeper.rb

Doorkeeper.configure do
  # Change the ORM that doorkeeper will use (requires ORM extensions install
  # Check the list of supported ORMs here: https://github.com/doorkeeper-gem
  orm :active_record

  # This block will be called to check whether the resource owner is authent
  # resource_owner_authenticator do
    # raise "Please configure doorkeeper resource_owner_authenticator block
    # Put your resource owner authentication logic here.
    # Example implementation:
    #   User.find_by(id: session[:user_id]) || redirect_to(new_user_session_
  # end

  resource_owner_from_credentials do |_routes|
    User.authenticate(params[:email], params[:password])
  end

  # ...
```

This will allow us to send the user email and password to the /oauth/token endpoint to authenticate user.

Then we need to implement the **authenticate** class method on the **app/models/user.rb** model file.

```
# app/models/user.rb
class User < ApplicationRecord
  # Include default devise modules. Others available are:
  # :confirmable, :lockable, :timeoutable, :trackable and :omniauthable
  devise :database_authenticatable, :registerable,
         :recoverable, :rememberable, :validatable

  validates :email, format: URI::MailTo::EMAIL_REGEXP

  # the authenticate method from devise documentation
  def self.authenticate(email, password)
    user = User.find_for_authentication(email: email)
    user&.valid_password?(password) ? user : nil
  end
end
```

You can read more on the authenticate method on Devise's github Wiki page.

Next, enable password grant flow in **config/initializers/doorkeeper.rb** , this will allow us to send the user email and password to the /oauth/token endpoint and get OAuth token in return.

```
Doorkeeper.configure do
  orm :active_record

  resource_owner_from_credentials do |_routes|
    User.authenticate(params[:email], params[:password])
  end
```

```
  # enable password grant
  grant_flows %w[password]

  # ....
```

You can search for “grant_flows” in this file, and uncomment and edit it.

Next, insert **allow_blank_redirect_uri true** into the configuration, so that we can create OAuth application with blank redirect URL (user won’t get redirected after login, as we are using API).

```
Doorkeeper.configure do
  orm :active_record

  resource_owner_from_credentials do |_routes|
    User.authenticate(params[:email], params[:password])
  end

  grant_flows %w[password]

  allow_blank_redirect_uri true
  # ....
```

As the OAuth application we create is for our own use (not third part), we can skip authorization.

Insert **skip_authorization** into the configuration like this :

```
Doorkeeper.configure do
  orm :active_record

  resource_owner_from_credentials do |_routes|
    User.authenticate(params[:email], params[:password])
  end

  grant_flows %w[password]

  allow_blank_redirect_uri true

  skip_authorization do
    true
  end
  # ....
```

The authorization we skipped is something like this :

As we skipped authorization, user won’t need to click the “authorize” button to interact with our API.

Optionally, if you want to enable refresh token mechanism in OAuth, you can insert the **use_refresh_token** into the configuration. This would allow the client app to request a new access token using the refresh token when the current access token is expired.

```
Doorkeeper.configure do
  orm :active_record

  resource_owner_from_credentials do |_routes|
    User.authenticate(params[:email], params[:password])
  end

  grant_flows %w[password]

  allow_blank_redirect_uri true

  skip_authorization do
    true
  end

  use_refresh_token
  # ...
```

With this, we have finished configuring Doorkeeper authentication for our API.

Next, we will add the doorkeeper route in **routes.rb** , this will add the /oauth/* routes.

```
Rails.application.routes.draw do
  use_doorkeeper do
    skip_controllers :authorizations, :applications, :authorized_application
  end

  # ...
end
```

As we don't need the app authorization, we can skip the authorizations and authorized_applications controller. We can also skip the applications controller, as users won't be able to create or delete OAuth application.

Next, we need to create our own OAuth application manually in the console so we can use it for authentication.

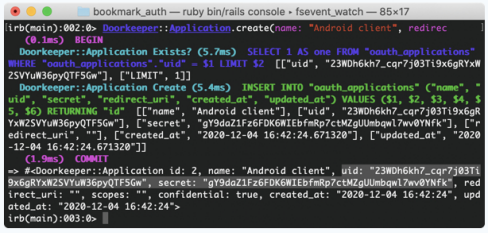
Create your own OAuth application

Open up rails console, `rails console`

Then create an OAuth application using this command :

```
Doorkeeper::Application.create(name: "Android client", redirect_uri: "", sco
```

You can change the name to any name you want, and leave redirect_uri and scopes blank.



This will create a record in the **oauth_applications** table. Keep note that the **uid** attribute and **secret** attribute, these are used for authentication on API later, uid = client_id and secret = client_secret.

For production use, you can create a database seed for initial creation of the OAuth applications in **db/seeds.rb** :

```
# db/seeds.rb

# if there is no OAuth application created, create them
if Doorkeeper::Application.count.zero?
  Doorkeeper::Application.create(name: "iOS client", redirect_uri: "", scope
  Doorkeeper::Application.create(name: "Android client", redirect_uri: "", s
  Doorkeeper::Application.create(name: "React", redirect_uri: "", scopes: ""
end
```

Then run `rake db:seed` to create these applications.

Doorkeeper::Application is just a namespaced model name for the **oauth_applications** table, you can perform ActiveRecord query as usual :

```
# client_id of the application
Doorkeeper::Application.find_by(name: "Android client").uid

# client_secret of the application
Doorkeeper::Application.find_by(name: "Android client").secret
```

Now we have Doorkeeper application set up, we can try to login user in the next section.

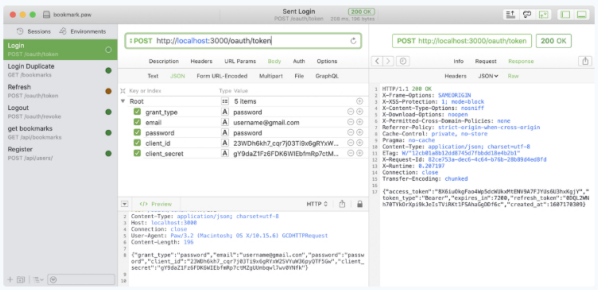
How to login , logout and refresh token using API

We will need a user created to be able to login / logout them using the OAuth endpoints, you can register a dummy user on the devise web UI if you haven't already (eg: localhost:3000/users/sign_up) or create one via the rails console.

The HTTP requests below can either send attributes using JSON format or URL-Encoded form.

Login

To login the user on the OAuth endpoint, we need to send a HTTP POST request to **/oauth/token**, with **grant_type**, **email**, **password**, **client_id** and **client_secret** attributes.



As we are using password in exchange for OAuth access and refresh token, the **grant_type** value should be **password**.

email and **password** is the login credential of the user.

client_id is the **uid** of the Doorkeeper::Application (OAuth application) we created earlier, with this we can identify which client the user has used to log in.

client_secret is the **secret** of the Doorkeeper::Application (OAuth application) we created earlier.

On successful login attempt, the API will return **access_token**, **refresh_token**, **token_type**, **expires_in** and **created_at** attributes.

We can then use **access_token** to call protected API that requires user authentication.

refresh_token can be used to generate and retrieve a new access token after the current access_token has expired.

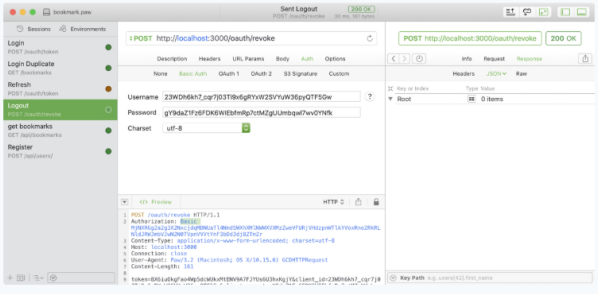
expires_in is the time until expiry for the access_token, starting from the UNIX timestamp of **created_at**, the default value is 7200 (seconds), which is around 2 hours.

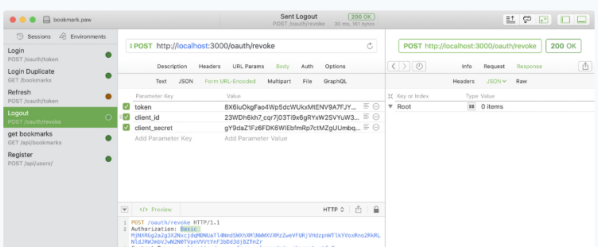
Logout

To log out a user, we can revoke the access token, so that the same access token cannot be used anymore.

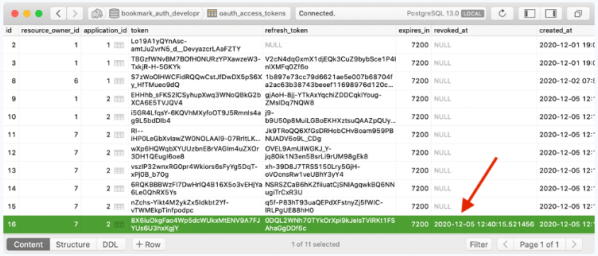
To revoke an access token, we need to send a HTTP POST request to **/oauth/revoke**, with **token**, **client_id** and **client_secret** attributes.

Other than these attributes, we also need to set Authorization header for the HTTP request to use Basic Auth, using **client_id** value for the **username** and **client_password** value for the **password**. (According to this reply in Doorkeeper gem repository)





After revoking a token, the token record will have a **revoked_at** column filled :



Refresh token

To retrieve a new access token when the current access token is (almost) expired, we can send a HTTP POST to **/oauth/token** , it is the same endpoint as login, but this time we are using “refresh_token” as the value for **grant_type**, and is sending the value of refresh token instead of login credentials.

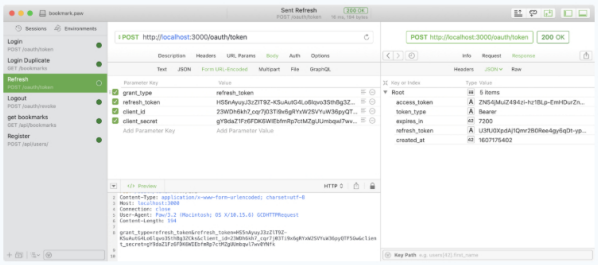
To refresh a token, we need to send **grant_type**, **refresh_token**, **client_id** and **client_secret** attributes.

grant_type needs to be equal to “**refresh_token**” here as we are using refresh token to authenticate.

refresh_token should be the refresh token value you have retrieved during login.

client_id is the **uid** of the Doorkeeper::Application (OAuth application) we created earlier.

client_secret is the **secret** of the Doorkeeper::Application (OAuth application) we created earlier.



On successful refresh attempt, the API return a new access_token and refresh_token, which we can use to call protected API that requires user authentication.

Create API controllers that require authentication

Now that we have user authentication set up, we can now create API controllers that require authentication.

For this, I recommend creating a base API application controller, then subclass this controller for controllers that require authentication.

Create a base API application controller (**application_controller.rb**) and place it in **app/controllers/api/application_controller.rb** .

```
# app/controllers/api/application_controller.rb
module Api
  class ApplicationController < ActionController::API
    # equivalent of authenticate_user! on devise, but this one will check th
    before_action :doorkeeper_authorize!

    private

    # helper method to access the current user from the token
    def current_user
      @current_user ||= User.find_by(id: doorkeeper_token[:resource_owner_id
    end
  end
end
```

The API application subclasses from ActionController::API, which is a lightweight version of ActionController::Base, and does not contain HTML layout and templating functionality (we dont need it for API anyway), and it doesn’t have CORS protection.

We add the **doorkeeper_authorize!** method in the before_action callback, as this will check if the user is authenticated with a valid token before calling methods in the controller, this is similar to the authenticate_user! method on devise.

We also add a **current_user** method to get the current user object, then we can attach the current user on some model's CRUD action.

As example of a protected API controller, let's create a bookmarks controller to retrieve all bookmarks.

app/controllers/api/bookmarks_controller.rb

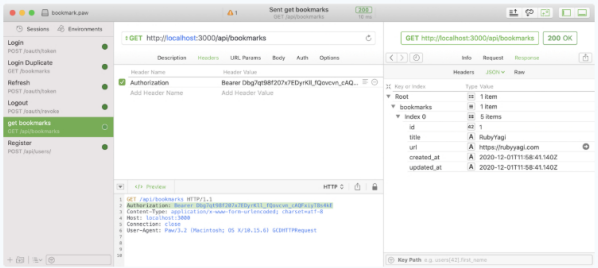
```
# app/controllers/api/bookmarks_controller.rb
module Api
  class BookmarksController < Api::ApplicationController
    def index
      @bookmarks = Bookmark.all
      render json: { bookmarks: @bookmarks }
    end
  end
end
```

The bookmarks controller will inherit from the base API application controller we created earlier, and it will return all the bookmarks in JSON format.

Don't forget to add the route for it in **routes.rb** :

```
Rails.application.routes.draw do
  # ....
  namespace :api do
    resources :bookmarks, only: %i[index]
  end
end
```

Then we can retrieve the bookmarks by sending a HTTP GET request to **/api/bookmarks**, with the user's access token in the Authorization Header (Authorization: Bearer [User Access Token])



To access protected API controllers, we will need to include the Authorization HTTP header, with the values of "Bearer [User Access Token]".

Create an endpoint for user registration

It would be weird if we only allow user registration through website, we would also need to add an API endpoint for user to register an account .

For this, let's create a users controller and place it in **app/controllers/api/users_controller.rb**.

The **create** action will create an user account from the supplied email and password.

```
module Api
  class UsersController < Api::ApplicationController
    skip_before_action :doorkeeper_authorize!, only: %i[create]

    def create
      user = User.new(email: user_params[:email], password: user_params[:pas

      client_app = Doorkeeper::Application.find_by(uid: params[:client_id])

      return render(json: { error: 'Invalid client ID'}, status: 403) unless

      if user.save
        # create access token for the user, so the user won't need to login
        access_token = Doorkeeper::AccessToken.create(
          resource_owner_id: user.id,
          application_id: client_app.id,
          refresh_token: generate_refresh_token,
          expires_in: Doorkeeper.configuration.access_token_expires_in.to_i,
          scopes: ''
```

```
        }

        # return json containing access token and refresh token
        # so that user won't need to call login API right after registration
        render(json: {
          user: {
            id: user.id,
            email: user.email,
            access_token: access_token.token,
            token_type: 'bearer',
            expires_in: access_token.expires_in,
            refresh_token: access_token.refresh_token,
            created_at: access_token.created_at.to_time.to_i
          }
        })
      else
        render(json: { error: user.errors.full_messages }, status: 422)
      end
    end

    private

    def user_params
      params.permit(:email, :password)
    end

    def generate_refresh_token
      loop do
        # generate a random token string and return it,
        # unless there is already another token with the same string
        token = SecureRandom.hex(32)
        break token unless Doorkeeper::AccessToken.exists?(refresh_token: to
      end
    end
  end
end
```

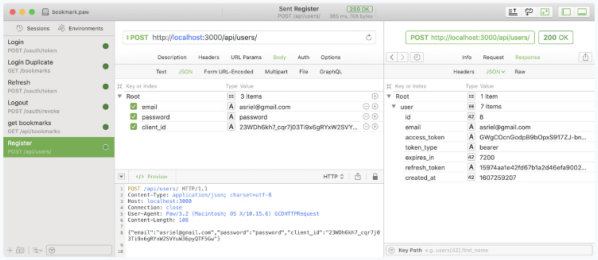
As the user doesn't have an account at this point, we want to exempt this action from requiring authentication information, so we added the line **skip_before_action :doorkeeper_authorize!, only: %i[create]** at the top. This will make the **create** method to skip running the **doorkeeper_authorize!** before_action method we defined in the base API controller, and the client app can call the user account creation API endpoint without authentication information.

We then create an AccessToken on successful user registration using **Doorkeeper::AccessToken.create()** and return it in the HTTP response, so the user won't need to login right after registration.

Remember to add a route for ths user registration action in **routes.rb** :

```
Rails.application.routes.draw do
  # ....
  namespace :api do
    resources :users, only: %i[create]
    resources :bookmarks, only: %i[index]
  end
end
```

Then we can call create user API by sending a HTTP POST request containing the user's **email**, **password** and **client_id** to **/api/users** like this :



The **client_id** is used to identify which client app the user is using for registration.

Revoke user token manually

Say if you suspect a user's access token has been misused or abused, you can revoke them manually using this function :

Doorkeeper::AccessToken.revoke_all_for(application_id, resource_owner)

application_id is the **id** of the Doorkeeper::Application (OAuth application) we want to revoke the user from.

resource_owner is the **user object**.

Example usage:

```
client_app = Doorkeeper::Application.find_by(name: 'Android client')
user = User.find(7)

Doorkeeper::AccessToken.revoke_all_for(client_app.id , user)
```

References

Devise wiki - How To: Find a user when you have their credentials

Doorkeeper wiki - Using Resource Owner Password Credentials flow

Jeff Davidson
5 years ago edited

Axel, Thanks for the great tutorial!!

0 0 Reply

Subscribe Privacy Do Not Sell My Data DISQUS