Does It Work?

635

Get started Log in

635

33% OFF an annual standard or premium subscription SAVE NOW

//offer/2021/q2-33-off)

Token-based Authentication with Ruby on Rails 5 API

Hristo Georgiev

Hristo Georgiev

Dec 15, 2018 · 17 Min read · 363,678 Views

Dec 15, 2018 · 17 Min read · 363,678 Views

Ruby Ruby on Rails

Introduction

With API-only applications so popular and Rails 5 right around the corner, the

Does It Work?

635

635

guide, I'll give a short overview of token-based authentication and how it is plemented into a Rails 5 API-only application.

What Is Token-based Authentication?

Token-based authentication (also known as JSON Web Token authentication) is a new way of handling the authentication of users in applications. It is an alternative to session-based authentication.

The most notable difference between the session-based and token-based authentication is that session-based authentication relies heavily on the server. A record is created for each logged-in user.

Token-based authentication is stateless - it does not store anything on the server but creates a unique encoded token that gets checked every time a request is made.

Unlike session-based authentication, a token approach would not associate a user with login information but with a unique token that is used to carry client-host transactions. Many applications, including Facebook, Google, and GitHub, use the token-based approach.

Does It Work?

635

635

nefits of Token-based Authentication

here are several benefits to using such an approach:

Cross-domain / CORS

Cookies and CORS don't mix well across different domains. A token-based approach allows you to make AJAX calls to any server, on any domain, because you use an HTTP header to transmit the user information.

Stateless

Tokens are stateless. There is no need to keep a session store since the token is a self-contained entity that stores all the user information in it.

Decoupling

You are no longer tied to a particular authentication scheme. Tokens may be generated anywhere, so the API can be called from anywhere with a single authenticated command rather than multiple authenticated calls.

Mobile Ready

Cookies are a problem when it comes to storing user information in native mobile applications. Adopting a token-based approach simplifies this saving process significantly.

635

cause the application does not rely on cookies for authentication, it is ulnerable to cross-site request attacks.

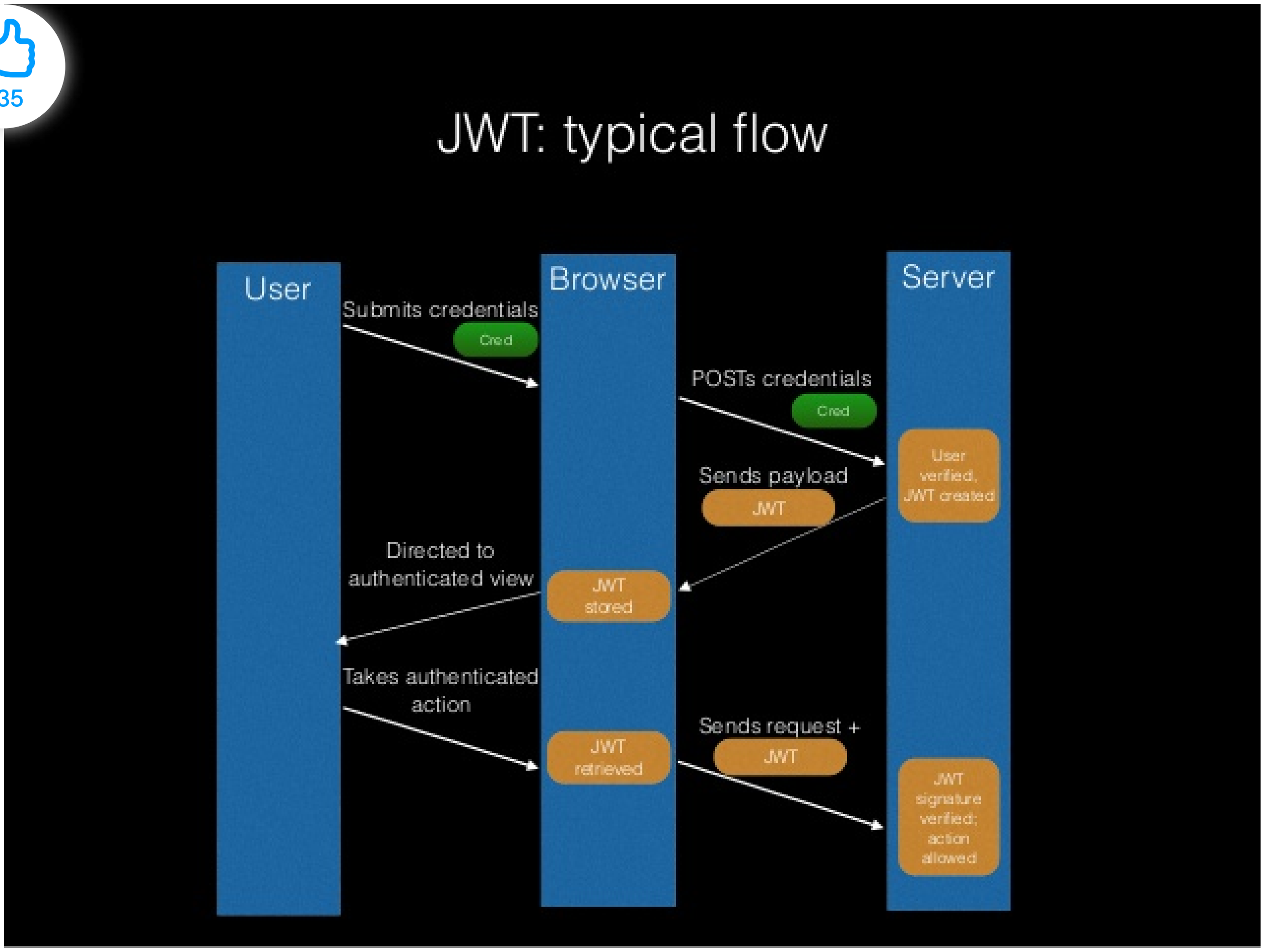
Performance

In terms of server-side load, a network roundtrip (e.g. finding a session on a database) is likely to take more time than calculating an HMACSHA256 code to validate a token and parsing its contents. This makes token-based authentication faster than the traditional alternative.

How Does Token-based Authentication Work?

The way token-based authentication works is simple. The user enters his or her credentials and sends a request to the server. If the credentials are correct, the server creates a unique HMACSHA256 encoded token, also known as JSON web token (JWT). The client stores the JWT and makes all subsequent requests to the server with the token attached. The server authenticates the user by comparing the JWT sent with the request to the one it has stored in the database. Here is a simple diagram of the process:

635



What Does a JWT Token Contain?

The token is separated into three base-64 encoded, dot-separated values. Each value represents a different type of data:

635

nsists of the type of the token (JWT) and the type of encryption algorithm 256) encoded in base-64.

Payload

The payload contains information about the user and his or her role. For example, the payload of the token can contain the e-mail and the password.

Signature

Signature is a unique key that identifies the service which creates the header. In this case, the signature of the token will be a base-64 encoded version of the Rails application's secret key (`Rails.application.secrets.secret_key_base`). Because each application has a unique base key, this secret key serves as the token signature.

Setting up a Token-based Authentication with Rails 5

Enough theory, it's time for practice. The first step is to create a new Rails 5 API-only application:

bash

```
rails _5.0.0.beta3_ new api-app --api
```

635

appending `--api` at the end of the generator, an API-only application will created. API-only applications are recent additions to the Rails platform. An API application is a trimmed-down version of standard Rails application without any of the unnecessary middleware, such as `.erb` views, helpers, and assets. API applications come with special middlewares such as `ActionController::API`, request throttling, easy CORS configuration and other custom-waived features for building APIs.

There are several requirements that need to be met before we can use the token-based approach:

- We need an accessible model.
- A way of encoding and decoding JWT tokens must be implemented.
- We need methods for checking if the user is authenticated.
- Controllers for creating and logging in users are also necessary.
- We need routes for creating users and logging them in and out.

Creating the User Model

First, the user model must be created:

bash
```
rails g model User name email password_digest
```

Does It Work?

635

635

bash
```bash
rails db:migrate
```

By running these methods, we create a user model with name, e-mail, and password fields and have its schema migrated in the database.

The method `has_secure_password` must be added to the model to make sure the password is properly encrypted into the database: `has_secure_password` is part of the `bcrypt` gem, so we have to install it first. Add it to the gemfile:

ruby
```ruby
#Gemfile.rb
gem 'bcrypt', '~> 3.1.7'
```

And install it:

bash
```bash
bundle install
```

With the gem installed, the method can be included in the model:

ruby
```ruby
#app/models/user.rb

class User < ApplicationRecord
```

Does It Work?

635

635

Encoding and Decoding JWT Tokens

Once the user model is done, the implementation of the JWT token generation can start. First, the `jwt` gem will make encoding and decoding of HMACSHA256 tokens available in the Rails application. First:

ruby

```ruby
gem 'jwt'
```

Then install it:

bash

```bash
bundle install
```

Once the gem is installed, it can be accessed through the `JWT` global variable. Because the methods that are going to be used to require encapsulation, a singleton class is a great way of wrapping the logic and using it in other constructs.

For those who are unfamiliar, **a singleton class** restricts the instantiation of a class to a single object, which comes in handy when only one object is needed

Does It Work?

635

635

b_token.rb

ruby

```ruby
class JsonWebToken
 class << self
   def encode(payload, exp = 24.hours.from_now)
     payload[:exp] = exp.to_i
     JWT.encode(payload, Rails.application.secrets.secret_key_base)
   end

   def decode(token)
     body = JWT.decode(token, Rails.application.secrets.secret_key_base)[0]
     HashWithIndifferentAccess.new body
   rescue
     nil
   end
 end
end
```

The first method, `encode`, takes three parameters – the user ID, the expiration time (1 day), and the unique base key of your Rails application – to create a unique token.

The second method, `decode`, takes the token and uses the application's secret key to decode it.

Here are the two cases in which these methods will be used:

Does It Work?

635

635

- For authenticating the user and generating a token for him/her using `encode`.
- To check if the user's token appended in each request is correct by using `decode`.

To make sure everything will work, the contents of the `lib` directory have to be included when the Rails application loads.

ruby

```ruby
    #config/application.rb
module ApiApp
  class Application < Rails::Application
    #.....
    config.autoload_paths << Rails.root.join('lib')
    #.....
    end
  end
```

Authenticating Users

Instead of using private controller methods, `simple_command` can be used. For more information about installation, check out the article simple_command.

The simple command gem is an easy way to create services. Its role is similar to the role of a helper, but it instead facilitates the connection between the

Does It Work?

635

635

troller and the model, rather than the controller and the view. In this way, we shorten the code in the models and controllers.

Add the gem to your `Gemfile`:

ruby

```ruby
gem 'simple_command'
```

And bundle it:

bash

```bash
bundle install
```

Then, the alias methods of the `simple_command` can be easily used in a class by writing `prepend SimpleCommand`. Here is how a command is structured:

ruby

```ruby
class AuthenticateUser
   prepend SimpleCommand

  def initialize()
   #this is where parameters are taken when the command is called
  end

  def call
   #this is where the result gets returned
  end

```

Does It Work?

635

635

command takes the user's e-mail and password then returns the user, if the credentials match. Here is how this can be done:

ruby

```ruby
# app/commands/authenticate_user.rb

class AuthenticateUser
  prepend SimpleCommand

  def initialize(email, password)
    @email = email
    @password = password
  end

  def call
    JsonWebToken.encode(user_id: user.id) if user
  end

  private

  attr_accessor :email, :password

  def user
    user = User.find_by_email(email)
    return user if user && user.authenticate(password)

    errors.add :user_authentication, 'invalid credentials'
    nil
  end
end
```

635

e command takes the parameters and initializes a class instance with `email` d `password` attributes that are accessible within the class. The private method `user` uses the credentials to check if the user exists in the database using `User.find_by_email`.

If the user is found, the method uses the built-in `authenticate` method. This method can be available by putting has_secure_password in the User model to check if the user's password is correct. If everything is true, the user will be returned. If not, the method will return `nil`.

Checking User Authorization

The token creation is done, but there is no way to check if a token that's been appended to a request is valid. The command for authorization has to take the `headers` of the request and decode the token using the `decode` method in the `JsonWebToken` singleton.

> *A refresher on headers:*
> *Http requests have fields known as headers. Headers can contain a wide variety of information about the request that can be helpful for the server interpreting the request. For example, a header can contain the format of the request body, authorization information, and other meta information*

Does It Work?

635

635

(you can find all the types here). Tokens are usually attached to the 'Authorization' header.

Here is how the code is structured:

ruby

```ruby
# app/commands/authorize_api_request.rb

class AuthorizeApiRequest
  prepend SimpleCommand

  def initialize(headers = {})
    @headers = headers
  end

  def call
    user
  end

  private

  attr_reader :headers

  def user
    @user ||= User.find(decoded_auth_token[:user_id]) if decoded_auth_token
    @user || errors.add(:token, 'Invalid token') && nil
  end

  def decoded_auth_token
    @decoded_auth_token ||= JsonWebToken.decode(http_auth_header)
  end
```

Does It Work?

635

635

```
      if headers['Authorization'].present?
        return headers['Authorization'].split(' ').last
      else
        errors.add(:token, 'Missing token')
      end
      nil
    end
  end
```

This code executes a chain of methods. **Let's start from the bottom and continue to the top.**

http_auth_header

The last method in the chain, `http_auth_header`, extracts the token from the authorization header received in the initialization of the class.

decoded_auth-token

The previous method in the chain is `decoded_auth_token`, which decodes the token received from `http_auth_header` and retrieves the user's ID.

user

The logic in the `user` method might seem abstract, so let's go through it line by line.

635

`coded_auth_token` returns false, `@user` will be `nil`.

Moving to the second line, the `user` method will either return the user or throw an error. In Ruby, the last line of the function is implicitly returned, so the command ends up returning the user object.

Implementing Helper Methods into the Controllers

All the logic for handling JWT tokens has been laid down. It is time to implement it in the controllers and put it to actual use. The two most essential pieces to implement are identifying user log-in and referencing the current user.

Login Users

First, let's start with the user's login:

ruby

```ruby
# app/controllers/authentication_controller.rb

class AuthenticationController < ApplicationController
 skip_before_action :authenticate_request

 def authenticate
   command = AuthenticateUser.call(params[:email], params[:passwor

   if command.success?
```

Does It Work?

635

635

```
        render json: { error: command.errors }, status: :unauthorized
      end
    end
  end
```

The `authenticate` action will take the JSON parameters for email and password through the `params` hash and pass them to the `AuthenticateUser` command. If the command succeeds, it will send the JWT token back to the user.

Let's put an endpoint for the action:

ruby

```ruby
#config/routes.rb
post 'authenticate', to: 'authentication#authenticate'
```

Authorizing Requests

To put the token to use, there must be a `current_user` method that will 'persist' the user. In order to have `current_user` available to all controllers, it has to be declared in the `ApplicationController`:

ruby

```ruby
#app/controllers/application_controller.rb
class ApplicationController < ActionController::API
 before_action :authenticate_request
```

Does It Work?

635

635

```
    private

    def authenticate_request
      @current_user = AuthorizeApiRequest.call(request.headers).resu
      render json: { error: 'Not Authorized' }, status: 401 unless @
    end
  end
```

By using `before_action`, the server passes the request headers (using the built-in object property `request.headers`) to `AuthorizeApiRequest` every time the user makes a request. Calling `result` on `AuthorizeApiRequest.call(request.headers)` is coming from `SimpleCommand` module where it is defined as `attr_reader :result`. The request results are returned to the `@current_user`, thus becoming available to all controllers inheriting from `ApplicationController`.

Does It Work?

Let's see how everything works. Start the Rails console in the application's root directory:

bash

```bash
rails c
```

Does It Work?

635

635

ate a user and insert it into the console:

bash

```
User.create!(email: 'example@mail.com' , password: '123123123' , password_c
```

To see how authorization works, there needs to be a resource that has to be requested. Let's scaffold a resource. In your terminal, run:

bash

```
rails g scaffold Item name:string description:text
```

This will create a resource named `Item` from top to bottom – a model, a controller, routes, and views. Migrate the database:

bash

```
rails db:migrate
```

Now, start the server and use cURL to post the credentials to `localhost:3000/authenticate`. Here is how the request should look:

Highlight

bash

```
xample@mail.com","password":"123123123"}' http://localhost:3000/authenticate
```

Your token will now be returned

Does It Work?

635

635

bash

```
{"auth_token":"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoxL
```

Great! A token has been generated. Let's check if the resource is reachable. You can do it by making a `GET` request to `localhost:3000/items`:

bash

```
$ curl http://localhost:3000/items
{"error":"Not Authorized"}
```

The resource is not reachable because the token has not been prepended to the headers of the request. Copy the previously generated token and put it in the `Authorization` header:

bash

```
DZ9.xsSwcPC22IR71OBv6bU_OGCSyfE89DvEzWfDU0iybMA" http://localhost:3000/items
```

With the token prepended, an empty array (`[]`) is returned. This is normal – after you add any items, you will see them returned in the request.

Awesome! Everything works.

If you missed something, the project has been uploaded on GitHub. If you have

Does It Work?

635

635

LEARN MORE

SOLUTIONS

Pluralsight Skills (/product/skills)

Pluralsight Flow (/product/flow)

Government (/industries/government)

Gift of Pluralsight (/gift-of-pluralsight)

View Pricing (/pricing)

Contact Sales (/product/contact-sales)

Skill up for free (/product/skills/free)

Does It Work?
Browse library (/browse)

635

Role IQ (/product/role-iq)

635