<script type="module"> import mermaid from 'https://cdn.jsdelivr.net/npm/mermaid@10/dist/mermaid.esm.min.mjs'; mermaid.initialize({ startOnLoad: true, theme: 'dark' }); </script>

Backend Authentication ด้วย Ruby on Rails

26 เมษายน 2567

Overview

- Ruby on Rails
- Authentication and Authorization
- HTTP Basic Access Authenentication
- Username with Secured Password
  - has_secured_password
  - Session
  - Cookie
- Token
  - JWT Token
  - One Time Password
- Authorization with OAuth

Ruby on Rails

- MVC Web Framework
- Convention over Configuration

```
// models
user.rb

// views
users/index.html
users/show.html

// controllers
users_controller.rb

// services
user_service.rb
```

Backend Authentication

- การยืนยันตัวตน

```
current_user = User.find(id)

current_user.email
# => "john@example.com"
```

Authorization

- การยืนยันสิทธิ์ในการเข้าถึงข้อมูล

```
Post.all.count
# => 12032

current_user.posts.count
# => 23
```

- การยืนยันสิทธิ์ความเป็นเจ้าของ

```
if post.user == current_user
  post.update_attributes(post_params)

  # render status: 200 # ค่าเท่ากันกับคำสั่งข้างล่าง
  render status: :ok
else
  # render status: 403
  render status: :forbidden
end
```

HTTP Basic Access Authentication

RFC-7235

> A HTTP/1.1 authentication flow that use Message Syntax and Routing from `RFC-7230`, including the general framework previously described in `RFC-2617` and the related fields and status codes previously defined in `RFC-2616`.

HTTP Basic Access Authentication

 sequenceDiagram participant Client as Browser participant Server Client->>Server: Request to restricted URL Server-->>Client: 401 Unauthorized with header
WWW-Authenticate: Basic Client->>Client: Pop-up login prompt Client->>Server: Request with
Authorization: Basic BASE64_ENCODED_STRING Server-->>Client: Response Client->>Server: Subsequent requests with
Authorization: Basic BASE64_ENCODED_STRING

HTTP Basic Access Authentication

> Encode Username and Password in Base64 encoded string then include that in every HTTP requests.

HTTP Basic Access Authentication

HTTP Basic Access Authentication

https://api.rubyonrails.org/classes/ActionController/HttpAuthentication/Basic.html

```ruby
# app/controllers/posts_controller.rb
class PostsController < ApplicationController
  http_basic_authenticate_with name: "admin",
                               password: "password",
                               only: :edit

  def index
    render plain: "Everyone can see list of Posts"
  end

  def show
    render plain: "Everyone can see Post's information"
  end

  def edit
    render plain: "Only people who know the password can edit"
  end
end
```

Username with Secured Password

```ruby
# Schema: User(name:string, password_digest:string, recovery_password_digest:string)
class User < ActiveRecord::Base
  has_secure_password
  has_secure_password :recovery_password, validations: false
end
```

Username with Secured Password

ActiveModel::SecurePassword

```ruby
user = User.new(name: "david", password: "", password_confirmation: "nomatch")

user.save                                                  # => false, password required
user.password = "vr00m"
user.save                                                  # => false, confirmation doesn't match
user.password_confirmation = "vr00m"
user.save                                                  # => true

user.authenticate("notright")                              # => false
user.authenticate("vr00m")                                 # => user
User.find_by(name: "david")&.authenticate("notright")      # => false
User.find_by(name: "david")&.authenticate("vr00m")         # => user

user.recovery_password = "42password"
user.recovery_password_digest                              # => "$2a$04$iOfhwahFymCs5weB3BNH/uXkTG65HR.qpW.bNhEjFP3ftli3o5DQC"
user.save                                                  # => true

user.authenticate_recovery_password("42password")          # => user

user.update(password: "pwn3d", password_challenge: "")     # => false, challenge doesn't authenticate
user.update(password: "nohack4u", password_challenge: "vr00m") # => true

user.authenticate("vr00m")                                 # => false, old password
user.authenticate("nohack4u")                              # => user
```

Guest Account

```ruby
class Account
  include ActiveModel::SecurePassword

  attr_accessor :is_guest, :password_digest

  has_secure_password

  def errors
    super.tap { |errors| errors.delete(:password, :blank) if is_guest }
  end
end

account = Account.new
account.valid? # => false, password required

account.is_guest = true
account.valid? # => true
```

Session

 sequenceDiagram participant Client participant Server Client->>Server: Authenticate with Server Server->>Server: Generate session ID Server->>Server: Keep information in session storage Server-->>Client: Response with session ID Client->>Server: Request with session ID Server->>Server: Retrieve data in session storage Server-->>Client: Response with requested data

Session

```ruby
# app/controllers/sessions_controller.rb
def create
  # ...
  session[:current_user_id] = @user.id
  # ...
end
```

```ruby
# app/controllers/posts_controller.rb
def index
  current_user = User.find_by_id(session[:current_user_id])
  posts = current_user.posts
  # ...
end
```

Cookie

 sequenceDiagram participant Client participant Server Client->>Server: Authenticate with Server Server->>Server: Generate session ID Server->>Server: Encrypt session data Server->>Server: Set session ID as cookie Server-->>Client: Response with cookie Client->>Server: Request with cookie Server->>Server: Decrypt session data in cookie Server-->>Client: Response with requested data

Cookie

```ruby
# Sets a simple session cookie.
# This cookie will be deleted when the user's browser is closed.
cookies[:user_name] = "david"

# Cookie values are String-based. Other data types need to be serialized.
cookies[:lat_lon] = JSON.generate([47.68, -122.37])

# Sets a cookie that expires in 1 hour.
cookies[:login] = { value: "XJ-122", expires: 1.hour }

# Sets a cookie that expires at a specific time.
cookies[:login] = { value: "XJ-122", expires: Time.utc(2020, 10, 15, 5) }

# Sets a signed cookie, which prevents users from tampering with its value.
# It can be read using the signed method `cookies.signed[:name]`
cookies.signed[:user_id] = current_user.id

# Sets an encrypted cookie value before sending it to the client which
# prevent users from reading and tampering with its value.
# It can be read using the encrypted method `cookies.encrypted[:name]`
cookies.encrypted[:discount] = 45

# Sets a "permanent" cookie (which expires in 20 years from now).
cookies.permanent[:login] = "XJ-122"

# You can also chain these methods:
cookies.signed.permanent[:login] = "XJ-122"
```

Cookie

- You can only store about 4kb of data in a cookie.
- Cookies are sent along with every request you make.
- Rails sign cookie with `secret_key_base`. Keep it safe.

```
# Attacker can signed a cookie to be any user.
# If they have secret_key_base that Rails use to signed.
cookies.signed[:user_id] = current_user.id
```

JWT

JWT signatures are created using cryptographic algorithms, such as HMAC (Hash-based Message Authentication Code) or RSA (Rivest-Shamir-Adleman).

For HMAC:

```
A secret key is used to generate the signature.
The signature is a hash (e.g., SHA256) of the header and payload, encoded using the secret key.
```

For RSA:

```
A public/private key pair is used.
The signature is created using the private key and verified using the public key.
```

The signature ensures the integrity of the JWT and prevents tampering.

JWT Token

 sequenceDiagram participant Client participant Server Client->>Server: Request with JWT Server->>Server: Decode JWT Server->>Server: Verify JWT signature Server->>Server: Extract payload Server-->>Client: Response with requested data

JWT Token

```
# Gemfile
gem 'jwt'
```

```
# config/initializers/jwt.rb
JWT_SECRET = 'your_secret_key_here'
```

JWT Token

```ruby
# app/controllers/application_controller.rb
class ApplicationController < ActionController::Base
  before_action :authenticate_user!

  def authenticate_user!
    token = request.headers['Authorization']&.split(' ')&.last
    begin
      payload = JWT.decode(token, JWT_SECRET, true, algorithm: 'HS256')
      @current_user_id = payload[0]['user_id']
    rescue JWT::DecodeError
      render json: { error: 'Invalid token' }, status: :unauthorized
    end
  end

  def current_user
    @current_user ||= User.find(@current_user_id)
  end
end
```

JWT Token

```ruby
# app/controllers/users_controller.rb
class UsersController < ApplicationController
  def show
    user = current_user
    render json: { user: user }
  end
end
```

```ruby
# app/controllers/posts_controller.rb
class UsersController < ApplicationController
  def index
    posts = current_user.posts
    render json: { posts: posts }
  end
end
```

One Time Password

 sequenceDiagram participant User participant Server User->>Server: Request OTP Server->>Server: Generate OTP and store in database Server-->>User: Send OTP User->>Server: Provide OTP Server->>Server: Verify OTP Server-->>User: Response

One Time Password

```ruby
class User < ActiveRecord::Base
  has_secure_password

  generates_token_for :password_reset, expires_in: 15.minutes do
    # Last 10 characters of password salt, which changes when password is updated:
    password_salt&.last(10)
  end
end

user = User.first

token = user.generate_token_for(:password_reset)
User.find_by_token_for(:password_reset, token) # => user
# 16 minutes later...
User.find_by_token_for(:password_reset, token) # => nil

token = user.generate_token_for(:password_reset)
User.find_by_token_for(:password_reset, token) # => user
user.update!(password: "new password")
User.find_by_token_for(:password_reset, token) # => nil
```

Authorization with OAuth

 sequenceDiagram participant User participant Client participant Authorization_Server participant Resource_Server User->>Client: Clicks "Login with OAuth" Client->>Authorization_Server: Redirect to Authorization Endpoint Authorization_Server->>User: Login Page User->>Authorization_Server: Enters Credentials Authorization_Server->>Client: Authorization Code Client->>Authorization_Server: Requests Access Token Authorization_Server->>Client: Sends Access Token Client->>Resource_Server: Requests Protected Resource Resource_Server->>Client: Sends Protected Resource

Refresh Token

 sequenceDiagram participant Client participant Authorization_Server Client->>Authorization_Server: Request Access Token with Refresh Token Authorization_Server->>Client: Sends new Access Token

Q & A