rails g authentication から学ぶ Rails8.0 時代の認証

Shinichi Maeshima(@willnet)

Kaigi on Rails 2025

2025/09/27

Sponsor RubyStackNews

Reach senior Ruby & Rails engineers worldwide

Put your brand in front of experienced developers, tech leads, and decision makers across the global Ruby ecosystem.

Become a Sponsor

今日は 9 月 27 日ですね

なんの日でしょうか?

ホーム
話題を検索
通知
メッセージ
Grok
ブックマーク
コミュニティ
プレミアム
認証済み組織
プロフィール
もっと見る
ポストする

willnet

3万 件のポスト

プロフィールを編集

willnet

@netwillnet

株式会社ウィルネット代表。Railsを利用している会社の技術顧問をしています。ginza.rbとclean-rails.orgの主催。著書にパーフェクトRuby on Rails(共著)。問い合わせは willnet.jp からお願いします。

tokyo willnet.jp お誕生日おめでとうございます。
2007年6月からXを利用しています

998 フォロー中 **3,134** フォロワー

まだ認証されていません

認証を受けると、返信の強化、アナリティクス、広告のないブラウジングなどを利用できます。今すぐプロフィールをアップグレードしましょう。

認証される

ポスト 返信 ハイライト 記事 メディア いいね

検索

おすすめ

新卒えんじにゃ
@engn_nnnaa
フォロー

経理のおしし@iCARE
@iCARE_Oshishi
フォロー

スタメン デザイナー...
@stmn_designer
フォロー

さらに表示

「いま」を見つけよう

日本のトレンド
#HAPPY_KAI_DAY
2,197件のポスト

日本のトレンド
諸子百苑

日本のトレンド
#髙橋未来虹生誕祭
1,725件のポスト

人気の画像・トレンド
#HAPPY_DOHA_DAY
1,539件のポスト

さらに表示

誕生日でした 

Shinichi Maeshima

Willnet Inc.

宙 @netwillnet

𝕋 @willnet

 https://blog.willnet.in

ginza.rb やってます

willnet

Contact

"つくれる"のその先へ

つくるときの障害を取りのぞき
楽しく継続性のあるWebサービス開発を実現します

Top
About
Clients
Service
Profile

技術顧問業をしています

We’re not hiring!

顧問先は週１程度
空きあります

business-details Public

Edit Pins Watch 0 Fork 0 Star

main 1 Branch 0 Tags Go to file t Add file Code

willnet 登壇内容の再演について追加 88c3d2a · 8 months ago 3 Commits

README.md 登壇内容の再演について追加 8 months ago

README

株式会社ウィルネットとして提供する技術顧問サービスについて

これはなに

株式会社ウィルネットとして提供する技術顧問サービスの内訳です。「技術顧問」という単語はかなり曖昧であり、お手伝い先に対して具体的に何を提供するのかがわかりづらいという問題があります。株式会社ウィルネット代表である @willnet は2016年1月から技術顧問サービスを始めており、この文章の執筆時点 [1] でおよそ9年のキャリアがあります。その間やってきたことをまとめることで、Rails関連で困っている会社が弊社に依頼をする内容を具体的

About

株式会社ウィルネットとして提供
術顧問サービスについて

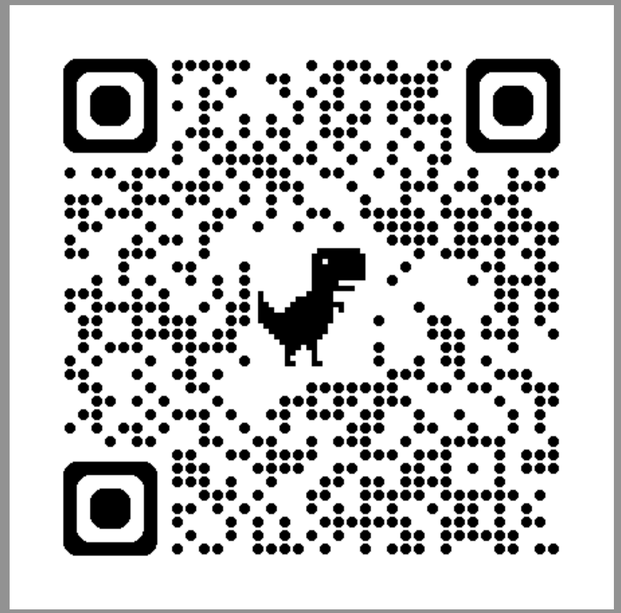
willnet.jp

Readme
Activity
Custom properties
1 star
0 watching
0 forks
Report repository

Releases

No releases published
Create a new release

Packages



今日は認証の話をします

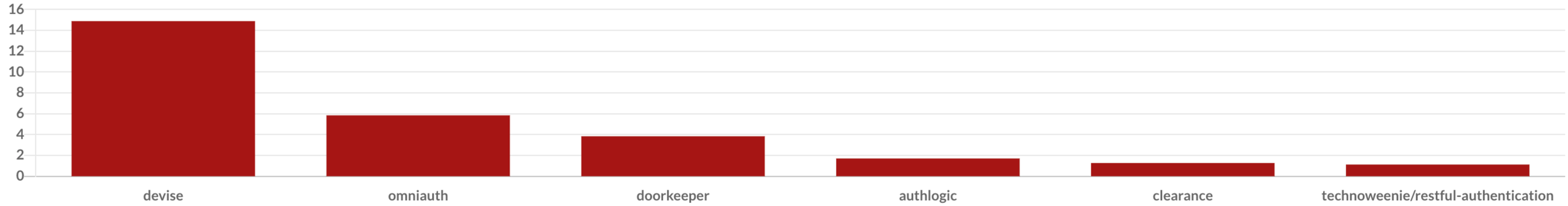
Rails で認証といえば

CATEGORY

Web Authentication

Edit this category

Handle the authentication of users in web applications written in frameworks like Rails



devise はダントツ

Full | Compact | Table | Order by **Project Score**

devise

14.89

No release in over a year

devise | heartcombo/devise | Homepage | Documentation | Source Code | Bug Tracker | Wiki

Flexible authentication solution for Rails with Warden

devise の特徴

- warden の上に作られている認証ライブラリ
  - warden は Rack アプリケーション向けに認証機能を追加するライブラリ
- モジュールを追加すると機能が増える (Confirmable 、 Recoverable など)
- devise を更に拡張したライブラリが数多くある
  - 2 要素認証 (devise-two-factor) 、メール招待 (deviseinvitable) など
- 利用例やカスタマイズ例が豊富
- おそらく仕事で認証ライブラリをいれる、となったときに一番最初に候補に上がる gem

devise は初心者にオススメできない

- 公式 README に注意書きがある
- > If you are building your first Rails application, we recommend you do not use Devise. Devise requires a good understanding of the Rails Framework. In such cases, we advise you to start a simple authentication system from scratch
  - もしあなたが初めて Rails アプリケーションを作るのであれば、 Devise を使わないことをおすすめします。 Devise を使うには Rails フレームワークについて十分な理解が必要です。そのような場合には、ゼロからシンプルな認証システムを作ることをおすすめします (ChatGPT 訳)
- devise の振る舞いを変えたい、とか調査をしたいときに rack, warden, devise, rails のコードベースを行ったり来たりする必要があり大変
- 各社、限られた devise 職人だけが devise のカスタマイズを担当することになりがち

とはいえ devise をみんな使う日々

巷に数多くある他の認証用 gem も devise の牙城を崩す気配は今のところ見えない (個人の感想です)

そんな中 Rails8.0 で
rails g authentication が使えるようになった

なぜいまになって認証ジェネレータが追加されたのか

“Rails now include all the key building blocks needed to do basic authentication, but many new developers are still uncertain of how to put them together, so they end up leaning on all-in-one gems that hide the mechanics. While these gems are great, and many people enjoy using them, they should not be seen as a necessity. We can teach Rails developers how to use the basic blocks by adding a basic authentication generator that essentially works as a scaffold, but for authentication.”

なぜいまになって認証ジェネレータが追加されたのか (ChatGPT 訳)

“ Rails には、基本的な認証を実装するために必要な部品はすべて揃っています。しかし、多くの初心者開発者はそれらをどう組み合わせればよいのか分からず、仕組みが隠されたオールインワンの gem に頼ってしまうことがよくあります。こうした gem は便利で多くの人に使われていますが、必須のものと考える必要はありません。 Rails 開発者には、基本的な部品を組み合わせて使う方法を教えられます。そのために「認証用のジェネレータ」を追加し、認証のための雛形（ scaffold ）のように使えるようにすればよいのです。”

なぜいまになって認証ジェネレータが追加されたのか (補足 1)

- そもそも Rails は認証用のメソッド (has_secure_password) を昔 (Rails >=3.1) から備えていたけれど、部品だけの形で提供されているためどのように使うかがわかりづらい
- 結果として gem を入れて README の通りに設定すれば OK な gem(例 : devise) が選択されることが多かった
  - が、先程話したように devise は初心者にはおすすめできない
- 認証用のメソッドをどのように組み合わせるとよいか道を示すためのツールとして認証ジェネレータを作った

なぜいまになって認証ジェネレータが追加されたのか (補足 2)

- DHH が CTO を務める 37signals 社の製品に Campfire (https://once.com/campfire) がある
  - ここで認証機能のために使われたコードがベースとなっている (明言されているわけではない)
  - DHH が rails/rails に認証ジェネレータに関する Issue を立てた時期 (2023/12/27) と Campfire をリリースした時期 (2024/01/19) が近いので、おそらく Campfire のコードを使って認証ジェネレータを作れるな、と考えたのではないでしょうか (明言されているわけではない)
- Rails7.1~7.2 あたりで抽象度高く認証機能を作るための機能追加がされてきたというのもありそう

Rails8.0 からは認証ジェネレータを使えば OK？

- ではない
- 認証ジェネレータが提供している機能は現時点で以下のみ
  - メールアドレスとパスワードを利用したログイン
  - ログアウト
  - パスワードリセット

認証ジェネレータと devise の関係

- 認証ジェネレータはあくまでシンプルな認証機能の使い方を示すもの
  - すべての認証ユースケースを解決するためのものではない
- 今後数年は次のように棲み分けがされるのではないでしょうか
  - シンプルな認証機能だけで賄えるアプリケーションは認証ジェネレータ
  - 多様な認証機能が必要なアプリケーションは devise をはじめとしたサードパーティの gem

少し話題を変えます

認証はセキュリティに直結している

セキュリティの側面から認証を捉える必要がある

認証ジェネレータを使うリスク

- 認証ジェネレータで生成したコードに脆弱性が含まれていたときに対応するのがやや難しい
  - devise などの gem を利用していれば dependabot などで脆弱性に気づき、bundle update で対応できる
- もし Rails8.0 の提供しているものに限らず認証ジェネレータを利用する場合はジェネレータのアップグレード情報を注意深くウォッチする必要がある

カスタマイズに関するリスク

- 認証に関する既存のコードをカスタマイズするときにはセキュリティに関する知識と細心の注意が必要
- 「よくわからないけどこれをああしたら動いた」だとまずい
- devise などの gem をそのまま使うだけなら問題は少ないけど、なにかしらカスタマイズすることありますよね?

認証機能は◯◯◯を使えばOK、ではなく我々は認証やセキュリティについてできる限り知っておく必要がある

認証やセキュリティについてどうやって学ぶ?

- 書籍で学ぶ
  - 安全なウェブサイトの作り方 (https://www.ipa.go.jp/security/vuln/websecurity/about.html)
  - 体系的に学ぶ 安全な Web アプリケーションの作り方 第 2 版 (https://www.sbcr.jp/product/4797393163/)
- 認証ジェネレータから生成されるコードで学ぶ

認証やセキュリティについてどうやって学ぶ?

- 書籍で学ぶ
  - 安全なウェブサイトの作り方 (https://www.ipa.go.jp/security/vuln/websecurity/about.html)
  - 体系的に学ぶ 安全な Web アプリケーションの作り方 第 2 版 (https://www.sbcr.jp/product/4797393163/)
- 認証ジェネレータから生成されるコードで学ぶ

認証ジェネレータから生成されるコードで学ぶ

- 僕は 17 年前に restful-authentication(Rails1,2 向けの認証ジェネレータ) で認証の流れを学びました
- Rails8.0 の認証ジェネレータでも同様のことができるはず
  - ただし 17 年前と比べて高度に抽象化されているので解説なしだと難しい

解説します

- 前提 : Rails8.1.0.beta1

まず概要から

認証ジェネレータで生成される機能

- メールアドレスとパスワードを利用したログイン
- ログアウト
- パスワードリセット
- ユーザ登録はないので自分で作る必要がある

認証ジェネレータで生成されるモデル

- User
- Session
  - ユーザのログイン状態を管理するためのモデル
- Current
  - ログインユーザに Current.user でアクセスするための Current Attributes

認証ジェネレータで生成されるコントローラ

- sessionscontroller.rb
  - ログイン、ログアウト用のコントローラ
- passwordscontroller.rb
  - パスワードリセット用のコントローラ
- concerns/authentication.rb
  - 認証関連のヘルパメソッドが書かれているモジュール

```ruby
class SessionsController < ApplicationController
  allow_unauthenticated_access only: %i[ new create ]
  rate_limit to: 10, within: 3.minutes, only: :create,
    with: -> { redirect_to new_session_path, alert: "Try again later." }

  def new
  end

  def create
    if user = User.authenticate_by(params.permit(:email_address, :password))
      start_new_session_for user
      redirect_to after_authentication_url
    else
      redirect_to new_session_path, alert: "Try another email address or password."
    end
  end

  def destroy
    terminate_session
    redirect_to new_session_path, status: :see_other
  end
end
```

```ruby
class SessionsController < ApplicationController

  rate_limit to: 10, within: 3.minutes, only: :create,
    with: -> { redirect_to new_session_path, alert: "Try again later." }

  def new
  end

  def create
    if user = User.authenticate_by(params.permit(:email_address, :password))
      start_new_session_for user
      redirect_to after_authentication_url
    else
      redirect_to new_session_path, alert: "Try another email address or password."
    end
  end

  def destroy
    terminate_session
    redirect_to new_session_path, status: :see_other
  end
end
```

allowunauthenticatedaccess

- ログイン確認する beforeaction をスキップしている
- 直接 skipbeforeaction を書くのに慣れている人々からすると微妙だけど、
  初見の人に意味が伝わりやすいのはこっち

```
# concerns/authentication.rb
def allow_unauthenticated_access(**options)
  skip_before_action :require_authentication, **options
end
```

```ruby
class SessionsController < ApplicationController
  allow_unauthenticated_access only: %i[ new create ]

  def new
  end

  def create
    if user = User.authenticate_by(params.permit(:email_address, :password))
      start_new_session_for user
      redirect_to after_authentication_url
    else
      redirect_to new_session_path, alert: "Try another email address or password."
    end
  end

  def destroy
    terminate_session
    redirect_to new_session_path, status: :see_other
  end
end
```

ratelimit

- Rails7.2 から追加
- この書き方だと、同じ IP アドレスのクライアントから 3 分以内に 11 回以上ログインを試行するとそれ以降のアクセスは自動でログインページにリダイレクトされる
  - ブルートフォースアタックを防ぐのに利用している
- このレベルの要件であれば rack-attack いらずになってべんり

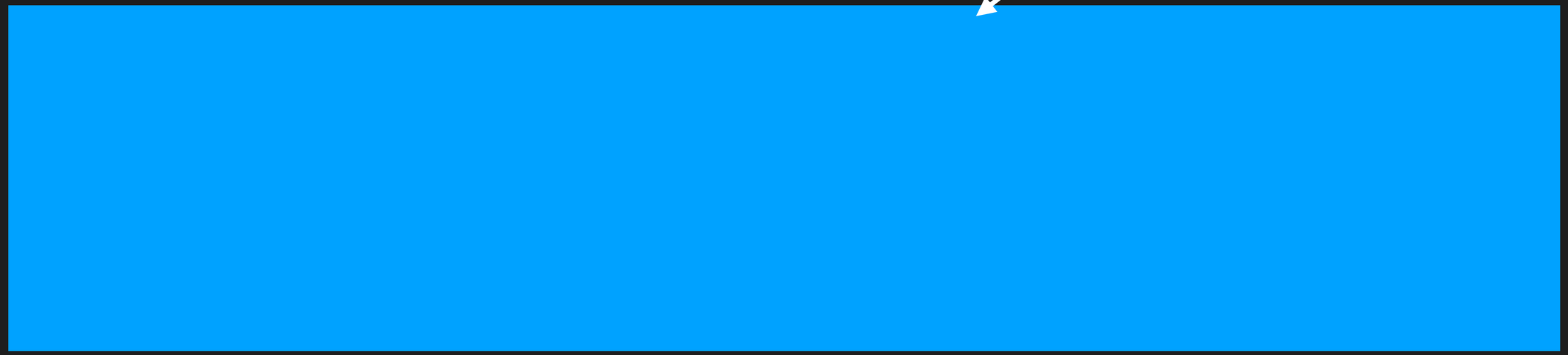
```
rate_limit to: 10, within: 3.minutes, only: :create,
  with: -> { redirect_to new_session_path, alert: "Try again later." }
```

```ruby
class SessionsController < ApplicationController
  allow_unauthenticated_access only: %i[ new create ]
  rate_limit to: 10, within: 3.minutes, only: :create,
    with: -> { redirect_to new_session_path, alert: "Try again later." }

  def create
    if user = User.authenticate_by(params.permit(:email_address, :password))
      start_new_session_for user
      redirect_to after_authentication_url
    else
      redirect_to new_session_path, alert: "Try another email address or password."
    end
  end

  def destroy
    terminate_session
    redirect_to new_session_path, status: :see_other
  end
end
```

ログインフォーム

```ruby
class SessionsController < ApplicationController
  allow_unauthenticated_access only: %i[ new create ]
  rate_limit to: 10, within: 3.minutes, only: :create,
    with: -> { redirect_to new_session_path, alert: "Try again later." }

  def new
  end

  def destroy
    terminate_session
    redirect_to new_session_path, status: :see_other
  end
end
```

認証用のアクション



```ruby
class SessionsController < ApplicationController
  allow_unauthenticated_access only: %i[ new create ]
  rate_limit to: 10, within: 3.minutes, only: :create,
    with: -> { redirect_to new_session_path, alert: "Try again later." }

  def new
  end

  def create
    if user =
      start_new_session_for user
      redirect_to after_authentication_url
    else
      redirect_to new_session_path, alert: "Try another email address or password."
    end
  end

  def destroy
    terminate_session
    redirect_to new_session_path, status: :see_other
  end
end
```

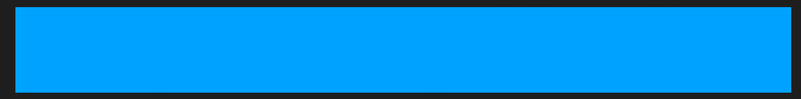
このメソッドどこから来た?

生成された User モデルのコード

```ruby
class User < ApplicationRecord
  has_secure_password
  has_many :sessions, dependent: :destroy

  normalizes :email_address, with: ->(e) { e.strip.downcase }
end
```

生成された User モデルのコード

ここで authenticateby が追加されている



```ruby
class User < ApplicationRecord

  has_many :sessions, dependent: :destroy

  normalizes :email_address, with: ->(e) { e.strip.downcase }
end
```

has_secure_password

- rails が提供する認証機能のメイン
- 主な機能
  - ユーザ作成時のパスワードを BCrypt でハッシュ化して格納
  - ユーザ作成時のパスワードのバリデーションを追加
  - ログイン時のパスワード比較用のメソッドを追加
  - パスワードリセット時のメールに含めるトークンの発行と検証用のメソッドを追加

has_secure_password - パスワードのハッシュ化

```ruby
user = User.new(password: 'password')
user.password_digest
#=> "$2a$12$0a23tjmHG3U/rsZ60wuETed7sug8QyVRfL4YCcRw5D83l1diiCaJe"
```

has_secure_password - BCrypt とは

- パスワードによく用いられるハッシュ関数
- 他のハッシュ関数よりも時間がかかる
  - ブルートフォースアタックがやりづらくなる
- キー拡張と呼ばれる処理を行う回数 (cost) を調整することでパスワード認証時間を増減できる
  - devise や bcrypt-ruby は 2 の 12 乗回 (cost 12) がデフォルト
  - gitlab は cost13
  - 昔から devise を利用しているひとはもっと少ない可能性があるので増やすのを推奨
- ソルトやレインボーテーブルについては省略

has_secure_password - パスワードのバリデーション

- 空文字だったらエラー
- 72 バイトより大きかったらエラー
  - BCrypt で使えるパスワードは 72 バイトまでという制約がある
- password_confirmation が存在して password と違ったらエラー
- 以上
- パスワードの複雑さなどをチェックしたい場合は自分で実装する

has_secure_password - ログイン時のパスワードの比較 (1)

- authenticate メソッドが用意されている

```
user = User.find_by(email: "willnet@example.com")
user.authenticate("password") #=> true
```

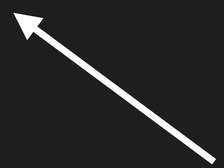
has_secure_password - ログイン時のパスワードの比較 (2)

```
User.authenticate_by(email: "willnet@example.com", password: "password") #=>  #<User:0x00000...
User.authenticate_by(email: "willnet@example.com", password: "wrong") #=> nil
```

- Rails7.1 以降は authenticate_by メソッドが使える
- 短いだけではなくてタイミングアタックを防止してくれる
  - 素朴な実装だと「メールアドレスまたはパスワードが間違えています」と表示してもレスポンス時間で登録済みメールアドレスかどうかの確認ができてしまう
  - authenticate_by はメールアドレスが DB にあってもなくても同じくらいの時間でレスポンスを返す

生成された User モデルのコード (再掲)

```ruby
class User < ApplicationRecord
  has_secure_password
  has_many :sessions, dependent: :destroy

end
```



これはなに?

normalizes

```ruby
normalizes :email_address, with: ->(e) { e.strip.downcase }
```

- 属性を正規化するメソッド (Rails7.1 から追加)
- 属性にアサインするタイミングと where などのクエリを実行するタイミングで lambda が実行される
- 入力したメールアドレスの前後にスペースが入ったり、大文字小文字が混在していても統一した形式で扱える
- メールアドレス以外にも電話番号とか郵便番号とかでべんりに使える

```ruby
class SessionsController < ApplicationController
  allow_unauthenticated_access only: %i[ new create ]
  rate_limit to: 10, within: 3.minutes, only: :create,
    with: -> { redirect_to new_session_path, alert: "Try again later." }

  def new
  end

  def create
    if user = User.authenticate_by(params.permit(:email_address, :password))

      redirect_to after_authentication_url
    else
      redirect_to new_session_path, alert: "Try another email address or password."
    end
  end

  def destroy
    terminate_session
    redirect_to new_session_path, status: :see_other
  end
end
```

ログインするためのメソッド

```ruby
# app/controllers/concerns/authentication.rb
def start_new_session_for(user)
  user.sessions.create!(
    user_agent: request.user_agent,
    ip_address: request.remote_ip
  ).tap do |session|
    Current.session = session
    cookies.signed.permanent[:session_id] =
      { value: session.id, httponly: true, same_site: :lax }
  end
end
```

```ruby
# app/controllers/concerns/authentication.rb
def start_new_session_for(user)
  user.sessions.create!(user_agent: request.user_agent, ip_address: request.remote_ip).tap do |session|
    Current.session = session
    cookies.signed.permanent[:session_id] =
      { value: session.id, httponly: true, same_site: :lax }
  end
end
```

User モデルと 1 対多関連の
Session モデルを create している

Session モデル

```
# id         :integer          not null, primary key
#  user_id    :integer          not null
#  ip_address :string
#  user_agent :string
#  created_at :datetime         not null
#  updated_at :datetime         not null
class Session < ApplicationRecord
  belongs_to :user
end
```

Session モデルは特別な実装を持っていない

Session モデルって何用途?

- User のログイン情報を管理している
  - これが DB 中にレコードとして存在する、というのがログイン状態を成立させる条件の一つになっている
- しかし巷では次のように User モデルの id を session に入れるコードがよく使われている
  - devise なども session を利用している

```
session[:user_id] = user.id
```

Session モデルって何用途?

- ログイン時の IP アドレスとユーザエージェントを保持している
  - 新しい環境からログインしたときに警告する、ができる
- 「ユーザが乗っ取られたときに今のログインセッションを消す」がやりやすい
  - cookies セッションだとブラウザ側にデータがあるので消せない
  - redis にセッションを格納すると消せるけど、 id は value 側にシリアライズした形で存在するので redis のデータを全件調べる必要があって遅い

```ruby
# app/controllers/concerns/authentication.rb
def start_new_session_for(user)
  user.sessions.create!(
    user_agent: request.user_agent,
    ip_address: request.remote_ip
  ).tap do |session|
    Current.session = session
    cookies.signed.permanent[:session_id] =
      { value: session.id, httponly: true, same_site: :lax }
  end
end
```

Current.session で Session モデルに
アクセスできるようにしている

Current モデルって何用途?

```ruby
class Current < ActiveSupport::CurrentAttributes
  attribute :session
  delegate :user, to: :session, allow_nil: true
end
```

Current モデルって何用途?

```
class Current <
  attribute :session
  delegate :user, to: :session, allow_nil: true
end
```

ActiveSupport::CurrentAttributes

- スレッド (もしくはファイバー) ごとに分かれていてリクエストのたびにリセットされるグローバル変数
- 例えば Current.user でどこからでもログインユーザの情報にアクセスできる
- 生成コードでは Current.session が truthy であればログイン済みとしている
- グローバル変数なので用法用量にお気をつけてご利用ください

```ruby
# app/controllers/concerns/authentication.rb
def start_new_session_for(user)
  user.sessions.create!(
    user_agent: request.user_agent,
    ip_address: request.remote_ip
  ).tap do |session|
    Current.session = session

    cookies.signed.permanent[:session_id] = { value: session.id, httponly: true, same_site: :lax }
  end
end
```

cookies に改ざん防止用トークンをつけた session.id を設定している

session ではなく cookies を利用しているのはなぜ?

- (さっきも書いたけど) 基本的には次のように session を利用することが多い

```
session[:user_id] = user.id
```

session ではなく cookies を利用しているのはなぜ?

- Action Cable で利用するため
- Action Cable は session を使えない
- 接続確立時の cookies にアクセスすることはできるので、 Action Cable を利用するときは cookies を使うのがお約束

認証ジェネレータは Action Cable 用のコードも生成している

```
# app/channels/application_cable/connection.rb
module ApplicationCable
  class Connection < ActionCable::Connection::Base
    identified_by :current_user

    def connect
      set_current_user || reject_unauthorized_connection
    end

    private
      def set_current_user
        if session = Session.find_by(id: cookies.signed[:session_id])
          self.current_user = session.user
        end
      end
  end
end
```

```ruby
class SessionsController < ApplicationController
  allow_unauthenticated_access only: %i[ new create ]
  rate_limit to: 10, within: 3.minutes, only: :create,
    with: -> { redirect_to new_session_path, alert: "Try again later." }

  def new
  end

  def create
    if user = User.authenticate_by(params.permit(:email_address, :password))
      start_new_session_for user

    else
      redirect_to new_session_path, alert: "Try another email address or password."
    end
  end

  def destroy
    terminate_session
    redirect_to new_session_path, status: :see_other
  end
end
```

ログイン後の URL に遷移する

ログイン後の URL

```
def after_authentication_url
  session.delete(:return_to_after_authenticating) || root_url
end
```

```ruby
module Authentication
  extend ActiveSupport::Concern

  included do
    before_action :require_authentication
  end

  private
    def require_authentication
      resume_session || request_authentication
    end

    def resume_session
      Current.session ||= find_session_by_cookie
    end

    def find_session_by_cookie
      Session.find_by(id: cookies.signed[:session_id]) if cookies.signed[:session_id]
    end

    def request_authentication
      session[:return_to_after_authenticating] = request.url
      redirect_to new_session_path
    end
end
```

```ruby
module Authentication
  extend ActiveSupport::Concern

  included do
    before_action :require_authentication
  end

  private
    def require_authentication
      resume_session || request_authentication
    end

    def resume_session
      Current.session ||= find_session_by_cookie
    end

    def find_session_by_cookie
      Session.find_by(id: cookies.signed[:session_id]) if cookies.signed[:session_id]
    end

    def request_authentication
      session[:return_to_after_authenticating] = request.url
      redirect_to new_session_path
    end
end
```

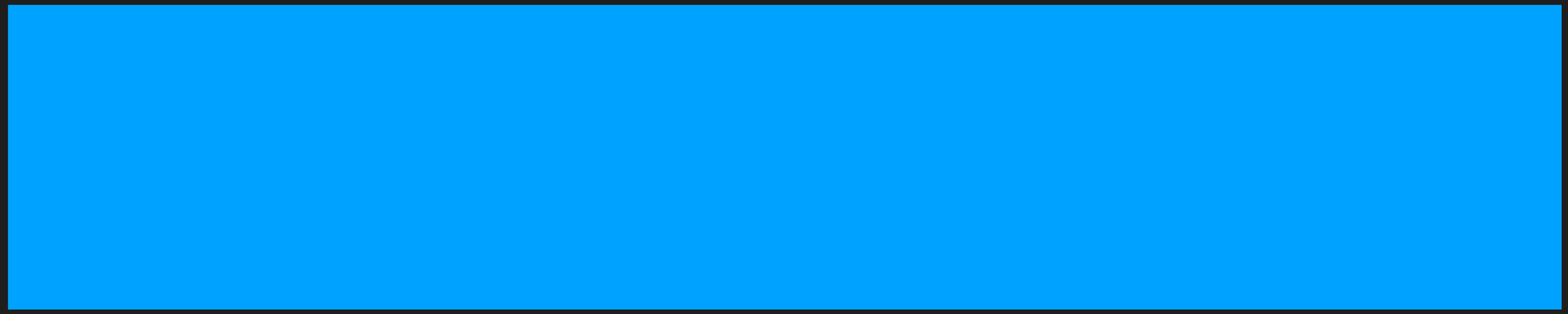
ログイン必須なアクションで実行されるフック

```ruby
module Authentication
  extend ActiveSupport::Concern

  included do
    before_action :require_authentication
  end

  private

    def resume_session
      Current.session ||= find_session_by_cookie
    end

    def find_session_by_cookie
      Session.find_by(id: cookies.signed[:session_id]) if cookies.signed[:session_id]
    end

    def request_authentication
      session[:return_to_after_authenticating] = request.url
      redirect_to new_session_path
    end
end
```

セッションの取得 (再開) を試みて
なければ認証をリクエストする

```ruby
module Authentication
  extend ActiveSupport::Concern

  included do
    before_action :require_authentication
  end

  private
    def require_authentication
      resume_session || request_authentication
    end

    def request_authentication
      session[:return_to_after_authenticating] = request.url
      redirect_to new_session_path
    end
end
```

cookies に有効な Session モデルの id が含まれていればログイン済み

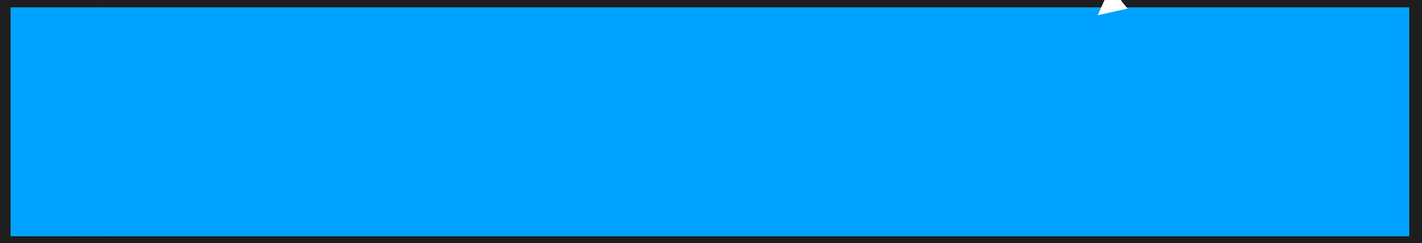


```ruby
module Authentication
  extend ActiveSupport::Concern

  included do
    before_action :require_authentication
  end

  private
    def require_authentication
      resume_session || request_authentication
    end

    def resume_session
      Current.session ||= find_session_by_cookie
    end

    def find_session_by_cookie
      Session.find_by(id: cookies.signed[:session_id]) if cookies.signed[:session_id]
    end

    [redacted]

end
```

未ログイン状態であればアクセスした
URL を session に一時保存して
ログインページにリダイレクト

```ruby
class SessionsController < ApplicationController
  allow_unauthenticated_access only: %i[ new create ]
  rate_limit to: 10, within: 3.minutes, only: :create,
    with: -> { redirect_to new_session_path, alert: "Try again later." }

  def new
  end

  def create
    if user = User.authenticate_by(params.permit(:email_address, :password))
      start_new_session_for user
      redirect_to after_authentication_url
    else
      redirect_to new_session_path, alert: "Try another email address or password."
    end
  end

  

end
```

ログアウト処理



```ruby
class SessionsController < ApplicationController
  allow_unauthenticated_access only: %i[ new create ]
  rate_limit to: 10, within: 3.minutes, only: :create,
    with: -> { redirect_to new_session_path, alert: "Try again later." }

  def new
  end

  def create
    if user = User.authenticate_by(params.permit(:email_address, :password))
      start_new_session_for user
      redirect_to after_authentication_url
    else
      redirect_to new_session_path, alert: "Try another email address or password."
    end
  end

  def destroy
    [redacted]
    redirect_to new_session_path, status: :see_other
  end
end
```

ログインセッションの削除

Session モデルを削除

```
def terminate_session
  
  cookies.delete(:session_id)
end
```

```
def terminate_session
  Current.session.destroy

end
```

cookies からも session.id を削除

ログイン・ログアウトについては
以上

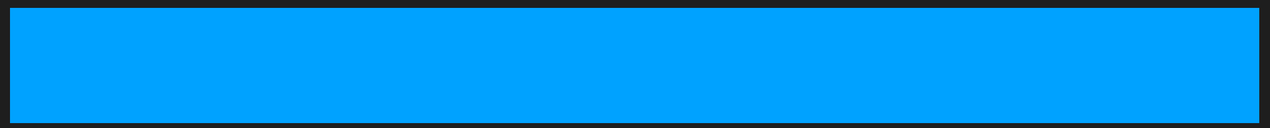
次はパスワードリセット

```ruby
class PasswordsController < ApplicationController
  allow_unauthenticated_access
  before_action :set_user_by_token, only: %i[ edit update ]
  rate_limit to: 10, within: 3.minutes, only: :create, with: -> { redirect_to new_password_path, alert: "Try again later." }

  def new
  end

  def create
    if user = User.find_by(email_address: params[:email_address])
      PasswordsMailer.reset(user).deliver_later
    end

    redirect_to new_session_path, notice: "Password reset instructions sent (if user with that email address exists)."
  end

  def edit
  end

  def update
    if @user.update(params.permit(:password, :password_confirmation))
      @user.sessions.destroy_all
      redirect_to new_session_path, notice: "Password has been reset."
    else
      redirect_to edit_password_path(params[:token]), alert: "Passwords did not match."
    end
  end

  private
    def set_user_by_token
      @user = User.find_by_password_reset_token!(params[:token])
    rescue ActiveSupport::MessageVerifier::InvalidSignature
      redirect_to new_password_path, alert: "Password reset link is invalid or has expired."
    end
end
```

文字が小さくなりすぎるので分割します

```ruby
class PasswordsController < ApplicationController
  allow_unauthenticated_acces
  rate_limit to: 10, within: 3.minutes, only: :create,
    with: -> { redirect_to new_password_path, alert: "Try again later." }

  def new
  end

  def create
    if user = User.find_by(email_address: params[:email_address])
      PasswordsMailer.reset(user).deliver_later
    end

    redirect_to new_session_path,
      notice: "Password reset instructions sent (if user with that email address exists)."
  end
end
```

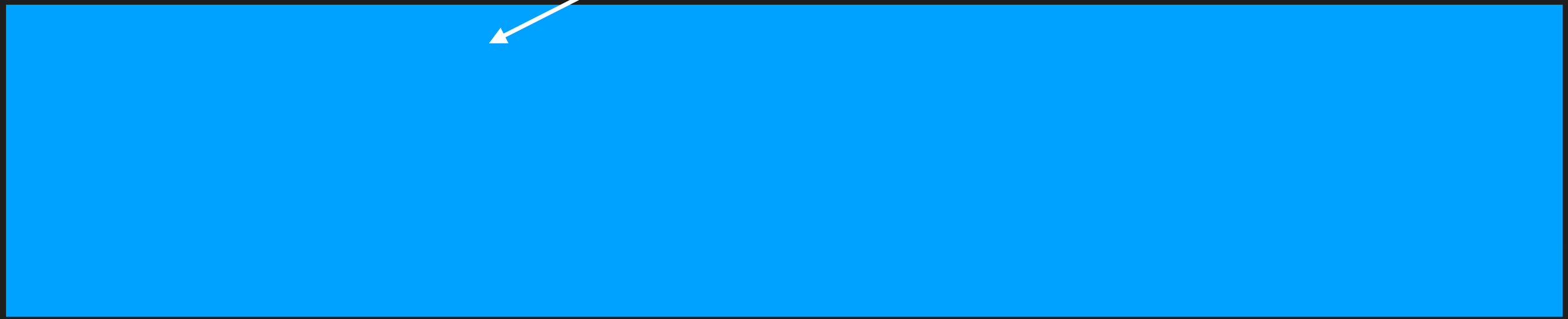
```ruby
class PasswordsController < ApplicationController
  allow_unauthenticated_acces
  rate_limit to: 10, within: 3.minutes, only: :create,
    with: -> { redirect_to new_password_path, alert: "Try again later." }

  def create
    if user = User.find_by(email_address: params[:email_address])
      PasswordsMailer.reset(user).deliver_later
    end

    redirect_to new_session_path,
      notice: "Password reset instructions sent (if user with that email address exists)."
  end
end
```



パスワードリセット用の
メールアドレスを入力する
フォーム

```ruby
class PasswordsController < ApplicationController
  allow_unauthenticated_acces
  rate_limit to: 10, within: 3.minutes, only: :create,
    with: -> { redirect_to new_password_path, alert: "Try again later." }

  def new
  end

end
```

対応するメールアドレスがあればパスワードリセット用のメールを送信する



```ruby
class PasswordsController < ApplicationController
  allow_unauthenticated_acces

  def new
  end

  def create
    if user = User.find_by(email_address: params[:email_address])
      PasswordsMailer.reset(user).deliver_later
    end

    redirect_to new_session_path,
      notice: "Password reset instructions sent (if user with that email address exists)."
  end
end
```

適当なメールアドレスにパスワードリセット用の
メールを送るいたずら防止

メールの文面 (HTML)

```
<p>
  You can reset your password within the next 15 minutes on
  <%= link_to "this password reset page", edit_password_url(@user.password_reset_token) %>.
</p>
```

reset.html.erb

パスワードリセット用のトークンを生成している

```
<p>
  You can reset your password within the next 15 minutes on
  <%= link_to "this password reset page", edit_password_url(                    %>.
</p>
```

reset.html.erb

has_secure_password - password_reset_token

- has_secure_password を宣言すると password_reset_token が使える (Rails8.0 から)
  - モデルに紐づいた 15 分を有効期間とするトークンを自動生成できる
  - 裏側では generates_token_for メソッド (Rails7.1 で追加) が使われてる
  - パスワードが変更されたらトークンは失効されるようになっている
  - 昔は DB にトークンを保存していたが、今では不要になってべんり

```
generates_token_for :"#{attribute}_reset", expires_in: 15.minutes do
  public_send(:"#{attribute}_salt")&.last(10)
end
```

```ruby
class PasswordsController < ApplicationController
  allow_unauthenticated_access
  before_action :set_user_by_token, only: %i[ edit update ]

  def edit
  end

  def update
    if @user.update(params.permit(:password, :password_confirmation))
      @user.sessions.destroy_all
      redirect_to new_session_path, notice: "Password has been reset."
    else
      redirect_to edit_password_path(params[:token]), alert: "Passwords did not match."
    end
  end

  private
    def set_user_by_token
      @user = User.find_by_password_reset_token!(params[:token])
    rescue ActiveSupport::MessageVerifier::InvalidSignature
      redirect_to new_password_path, alert: "Password reset link is invalid or has expired."
    end
end
```

```ruby
class PasswordsController < ApplicationController
  allow_unauthenticated_access

  def edit
  end

  def update
    if @user.update(params.permit(:password, :password_confirmation))
      @user.sessions.destroy_all
      redirect_to new_session_path, notice: "Password has been reset."
    else
      redirect_to edit_password_path(params[:token]), alert: "Passwords did not match."
    end
  end

  private
    def set_user_by_token
      @user = User.find_by_password_reset_token!(params[:token])
    rescue ActiveSupport::MessageVerifier::InvalidSignature
      redirect_to new_password_path, alert: "Password reset link is invalid or has expired."
    end
end
```

トークンが正しいかをチェック

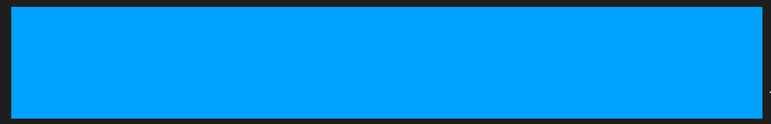
```ruby
class PasswordsController < ApplicationController
  allow_unauthenticated_access
  before_action :set_user_by_token, only: %i[ edit update ]

  def edit
  end

  def update
    if @user.update(params.permit(:password, :password_confirmation))
      @user.sessions.destroy_all
      redirect_to new_session_path, notice: "Password has been reset."
    else
      redirect_to edit_password_path(params[:token]), alert: "Passwords did not match."
    end
  end

  private
    def set_user_by_token
      @user = User.find_by_password_reset_token!(params[:token])
    rescue ActiveSupport::MessageVerifier::InvalidSignature
      redirect_to new_password_path, alert: "Password reset link is invalid or has expired."
    end
end
```

これも has_secure_password で生成される
メソッド

```ruby
class PasswordsController < ApplicationController
  allow_unauthenticated_access
  before_action :set_user_by_token, only: %i[ edit update ]

  def edit
  end

  def update
    if @user.update(params.permit(:password, :password_confirmation))
      @user.sessions.destroy_all
      redirect_to new_session_path, notice: "Password has been reset."
    else
      redirect_to edit_password_path(params[:token]), alert: "Passwords did not match."
    end
  end

  private
    def set_user_by_token
      @user = User.find_by_password_reset_token!(params[:token])
    rescue ActiveSupport::MessageVerifier::InvalidSignature

    end
end
```

トークンが不正だったらメールアドレス入力フォームに戻す

```ruby
class PasswordsController < ApplicationController
  allow_unauthenticated_access
  before_action :set_user_by_token, only: %i[ edit update ]

  def update
    if @user.update(params.permit(:password, :password_confirmation))
      @user.sessions.destroy_all
      redirect_to new_session_path, notice: "Password has been reset."
    else
      redirect_to edit_password_path(params[:token]), alert: "Passwords did not match."
    end
  end

  private
    def set_user_by_token
      @user = User.find_by_password_reset_token!(params[:token])
    rescue ActiveSupport::MessageVerifier::InvalidSignature
      redirect_to new_password_path, alert: "Password reset link is invalid or has expired."
    end
end
```

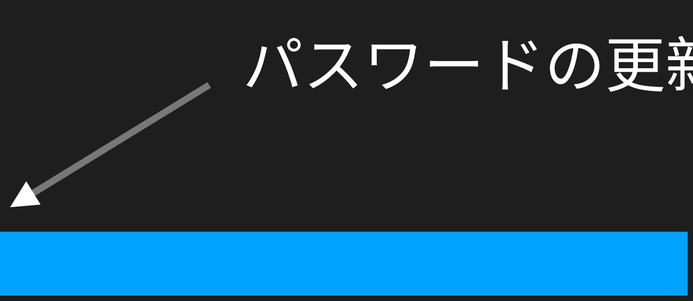


新しいパスワードを入力するフォーム

```ruby
class PasswordsController < ApplicationController
  allow_unauthenticated_access
  before_action :set_user_by_token, only: %i[ edit update ]

  def edit
  end

  private
    def set_user_by_token
      @user = User.find_by_password_reset_token!(params[:token])
    rescue ActiveSupport::MessageVerifier::InvalidSignature
      redirect_to new_password_path, alert: "Password reset link is invalid or has expired."
    end
end
```

新しいパスワードに更新する

```ruby
class PasswordsController < ApplicationController
  allow_unauthenticated_access
  before_action :set_user_by_token, only: %i[ edit update ]

  def edit
  end

  def update
    if @user.update(params.permit(:password, :password_confirmation))
      @user.sessions.destroy_all
      redirect_to new_session_path, notice: "Password has been reset."
    else
      redirect_to edit_password_path(params[:token]), alert: "Passwords did not match."
    end
  end

  private
    def set_user_by_token
      @user = User.find_by_password_reset_token!(params[:token])
    rescue ActiveSupport::MessageVerifier::InvalidSignature
      redirect_to new_password_path, alert: "Password reset link is invalid or has expired."
    end
end
```



```ruby
class PasswordsController < ApplicationController
  allow_unauthenticated_access
  before_action :set_user_by_token, only: %i[ edit update ]

  def edit
  end

  def update
    if @user.update(params.permit(:password, :password_confirmation))
      redirect_to new_session_path, notice: "Password has been reset."
    else
      redirect_to edit_password_path(params[:token]), alert: "Passwords did not match."
    end
  end

  private
    def set_user_by_token
      @user = User.find_by_password_reset_token!(params[:token])
    rescue ActiveSupport::MessageVerifier::InvalidSignature
      redirect_to new_password_path, alert: "Password reset link is invalid or has expired."
    end
end
```

パスワードを更新できたら該当ユーザのセッションを全部失効させる (セッションを乗っ取られたときにそのセッションを失効させる目的)

```ruby
class PasswordsController < ApplicationController
  allow_unauthenticated_access
  before_action :set_user_by_token, only: %i[ edit update ]

  def edit
  end

  def update
    if @user.update(params.permit(:password, :password_confirmation))
      @user.sessions.destroy_all
      redirect_to new_session_path, notice: "Password has been reset."
    else
      redirect_to edit_password_path(params[:token]), alert: "Passwords did not match."
    end
  end

  private
    def set_user_by_token
      @user = User.find_by_password_reset_token!(params[:token])
    rescue ActiveSupport::MessageVerifier::InvalidSignature
      redirect_to new_password_path, alert: "Password reset link is invalid or has expired."
    end
end
```

ログインページにリダイレクト

```ruby
class PasswordsController < ApplicationController
  allow_unauthenticated_access
  before_action :set_user_by_token, only: %i[ edit update ]

  def edit
  end

  def update
    if @user.update(params.permit(:password, :password_confirmation))
      @user.sessions.destroy_all
      redirect_to new_session_path, notice: "Password has been reset."
    else

    end
  end

  private
    def set_user_by_token
      @user = User.find_by_password_reset_token!(params[:token])
    rescue ActiveSupport::MessageVerifier::InvalidSignature
      redirect_to new_password_path, alert: "Password reset link is invalid or has expired."
    end
end
```

新しいパスワードと確認用のパスワードが一致してなかったらもう一回フォームに戻す

パスワードリセットについては
以上

認証ジェネレータが生成するコードから認証の流れについて解説してきました

素朴な流れは簡単でも細かく考え出すと多くの考慮すべきポイント(例：ログインセッションをどう持つのか) がある

知らないといけない知識もある
(例：タイミングアタック)

認証機能のセキュリティを担保するには我々の技術力向上及び最新情報のキャッチアップが不可欠

たいへんですが頑張ってやっていきましょう💪

ちょっと自信がないなあ、という方は…?

顧問先は週１程度
空きあります👀