

# Documentação API-BackEnd

## Projeto Controle de Acesso

---

### Resumo

1. Visão Geral do Projeto
2. Arquitetura do Projeto
3. Configuração do Ambiente de Desenvolvimento
4. Estrutura do Código
5. Modelo para Banco de Dados
6. Gemfile
7. Controladores
8. Rotas
9. Models

### Visão Geral do Projeto

#### Sobre o RubyOnRails:

**O que é Ruby on Rails:** Ruby on Rails, frequentemente abreviado como Rails, é um framework web escrito em Ruby que segue o princípio de convenção sobre configuração. Isso significa que, ao seguir certas convenções, desenvolvedores podem acelerar o desenvolvimento e manter uma estrutura organizada. A arquitetura de um projeto com Ruby on Rails é baseada no padrão arquitetural MVC (Model-View-Controller).

**Porque utilizamos ele:** Ao adotar o Ruby on Rails, os desenvolvedores se beneficiam de uma estrutura organizada que promove eficiência e consistência, facilitando o desenvolvimento e a manutenção de aplicações web robustas e escaláveis. A arquitetura baseada em MVC e as convenções do framework contribuem para a produtividade e a qualidade do código.

# Arquitetura do Projeto

## Modelo de Programação MVC:

**O que é MVC (Model view controller):** O Modelo de Programação MVC é um padrão arquitetural amplamente utilizado na construção de aplicativos de software, incluindo aplicações web. Ele divide uma aplicação em três componentes interconectados, cada um com responsabilidades distintas. O MVC visa melhorar a modularidade, escalabilidade e manutenibilidade do código.

## Divisões

### a. Modelo (Model)

O modelo representa a camada de dados da aplicação. Ele é responsável por gerenciar o acesso aos dados, realizar operações de leitura e escrita no banco de dados e fornecer os dados necessários para a aplicação. Os modelos também encapsulam a lógica de dados.

### b. Visão (View)

A visão trata da apresentação dos dados ao usuário. Ela exibe as informações do modelo de maneira formatada e interage com o usuário, coletando entrada quando necessário. As visões geralmente são responsáveis por estruturar a interface do usuário (UI) e podem ser atualizadas automaticamente quando o modelo é modificado.

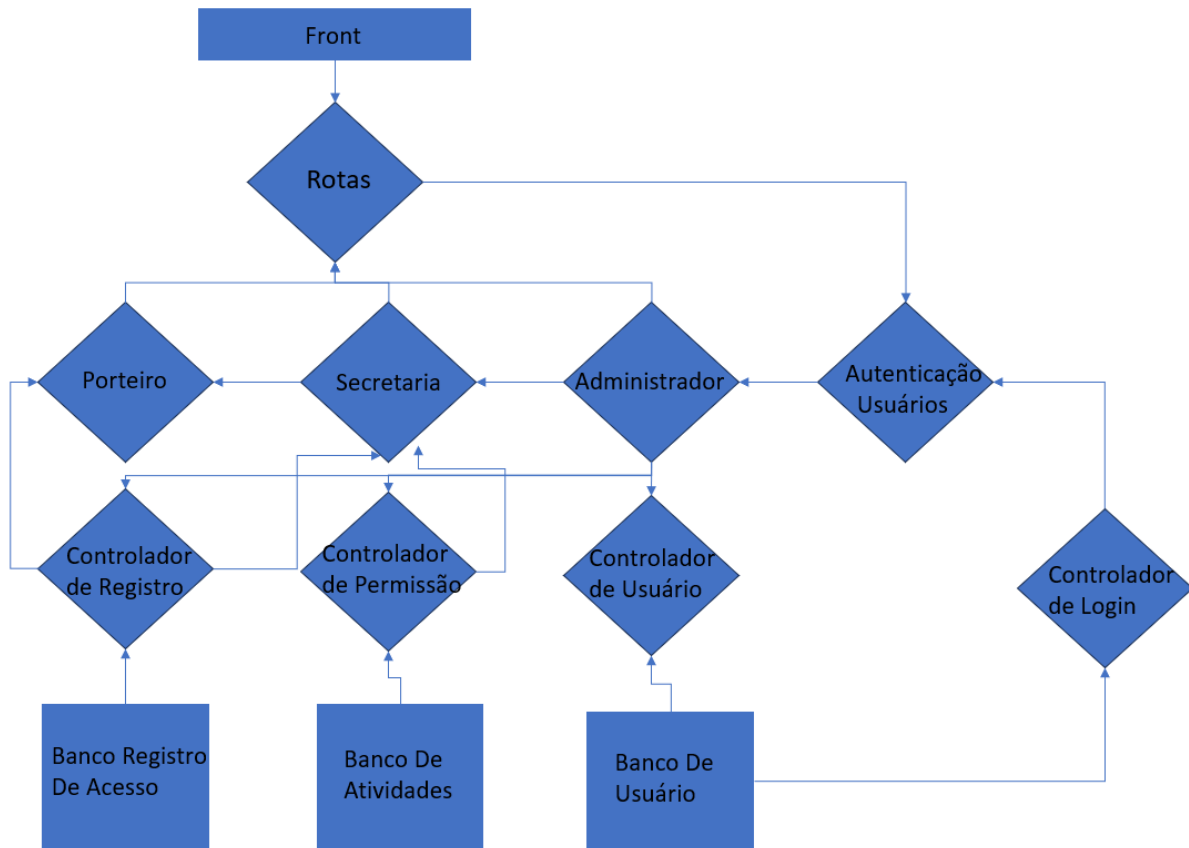
### e. Controlador (Controller)

O controlador atua como intermediário entre o modelo e a visão. Ele recebe as entradas do usuário, manipula as interações e atualiza o modelo conforme necessário. O controlador também é responsável por receber eventos da visão e decidir como responder a esses eventos. Ele age como um direcionador do fluxo de controle na aplicação.

### f. Conclusão

Ao seguir essa abordagem, a lógica de controle de acesso é distribuída entre os três componentes do MVC. O modelo gerencia as regras de permissão, a visão exibe elementos da interface do usuário com base nessas permissões e o controlador supervisiona a execução das ações, garantindo conformidade com as regras de acesso.

## Fluxograma Das Ferramentas:



## Configuração do Ambiente de Desenvolvimento

### Configurando Ferramentas do Sistema Ruby on Rails com MySQL:

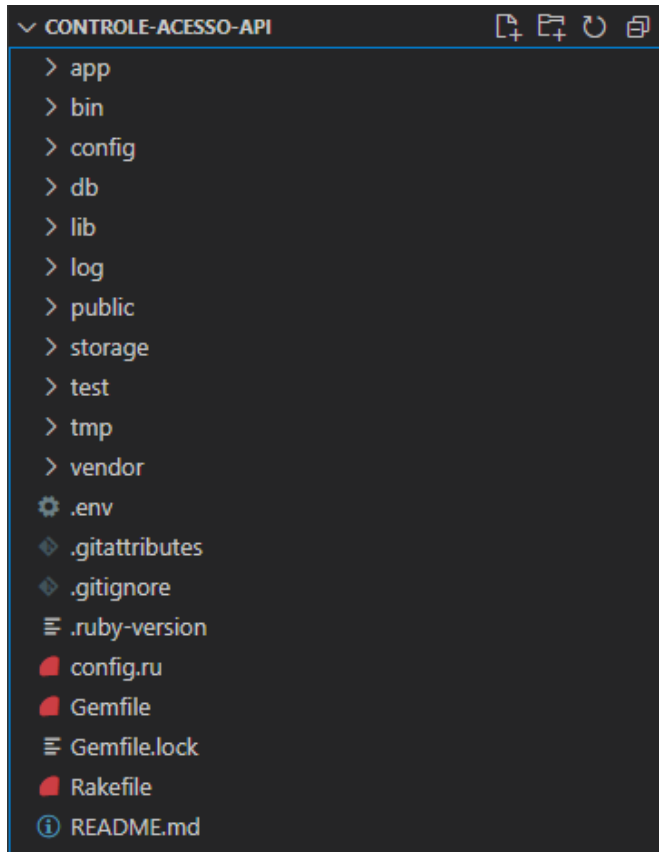
**Ferramentas do sistema:** Para começar a desenvolver uma aplicação Ruby on Rails com MySQL, você precisará instalar alguns componentes essenciais em seu ambiente de desenvolvimento. Abaixo estão os passos básicos para configurar o ambiente.

1. **Ruby:** Antes de instalar o Rails, é necessário ter o Ruby instalado. Você pode fazer o download da versão mais recente do Ruby em <https://www.ruby-lang.org/>. Algumas distribuições do sistema operacional já incluem o Ruby, então você pode verificar isso antes de fazer o download.
2. **RubyGems:** RubyGems é o sistema de gerenciamento de pacotes para Ruby. Geralmente, ele é incluído na instalação padrão do Ruby. Certifique-se de que está atualizado executando o seguinte no terminal ou prompt de comando: `gem update --system`.
3. **Rails:** O Rails é o framework Ruby para o desenvolvimento de aplicações web. Você pode instalá-lo usando o seguinte comando: `gem install rails`.
4. **MySQL:** O MySQL é um sistema de gerenciamento de banco de dados relacional. Faça o download e instale o MySQL a partir do site oficial: <https://www.mysql.com/>. Durante a instalação, configure o nome de usuário e senha do banco de dados, pois você precisará dessas informações mais tarde.
5. **Adapter do MySQL para Ruby (mysql2 gem):** O Rails requer um adaptador de banco de dados para se comunicar com o MySQL. O mysql2 é um adaptador popular que você pode instalar usando o seguinte comando: `gem install mysql2`.
6. **Execução do Migrations:** Após configurar o banco de dados no arquivo `database.yml`, você pode executar as migrações para criar as tabelas necessárias no banco de dados usando o seguinte comando: `rails db:migrate`.

**Resumo:** Com essas etapas, você deve ter um ambiente de desenvolvimento Ruby on Rails configurado para usar o MySQL como banco de dados. Lembre-se de consultar a documentação oficial do Ruby on Rails e do MySQL para obter informações mais detalhadas e atualizadas.

## Estrutura do Código

Organização do código:



### Pontos Importantes Organização de Pastas:

**App:** Em um projeto Ruby on Rails, a pasta "app" é fundamental, servindo como o núcleo que abriga grande parte do código-fonte da aplicação. Dentro dessa pasta, são encontradas subpastas específicas que desempenham papéis distintos.

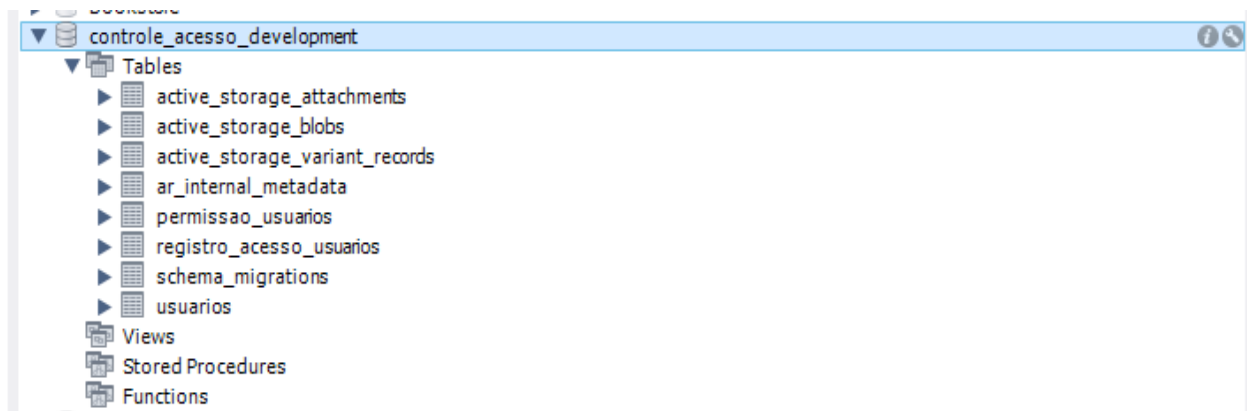
**Models:** A pasta app/models contém os modelos, representando a estrutura e a lógica de dados da aplicação, onde cada modelo geralmente corresponde a uma tabela no banco de dados. Em app/views, encontram-se as views, responsáveis pela apresentação dos dados ao usuário, utilizando HTML e a linguagem de template do Rails para gerar páginas dinâmicas.

**Controllers:** Os controllers, presentes em app/controllers, gerenciam as requisições do usuário, interagindo com modelos e views para processar e apresentar dados. Além dessas subpastas principais, a pasta "app" inclui diretórios como "assets" para arquivos estáticos (CSS, JavaScript) e "mailers" para lógica de envio de e-mails.

## Modelo para Banco de Dados

### Gerenciando o MySQL:

**Vantagens do MySQL:** A integração do MySQL com o Ruby on Rails oferece uma solução eficiente e escalável para o desenvolvimento de aplicações web. O suporte nativo do Rails facilita a configuração e uso do MySQL, conhecido por seu desempenho robusto e escalabilidade, tornando-o ideal para lidar com grandes volumes de dados e operações simultâneas. Sua estabilidade, comunidade ativa e recursos abundantes, juntamente com ferramentas de administração como o MySQL Workbench, proporcionam uma experiência de desenvolvimento confiável. Além disso, o MySQL oferece recursos avançados, incluindo suporte a transações ACID e flexibilidade para implementar lógica de banco de dados complexa, contribuindo para a integridade e eficácia do desenvolvimento Ruby on Rails.



### Tabelas Geradas por Rails:

**Divisão de Tabelas:** O próprio sistema do Rails cria tabelas adicionais para efetuar controles adicionais ao banco. Sendo elas active\_storage\_attachments, active\_storage\_attachments, active\_storage\_attachments e active\_storage\_attachments, schema\_migrations.

As tabelas **active\_storage\_attachments** e **active\_storage\_blobs** são criadas pelo Rails para gerenciar anexos de arquivos em uma aplicação. A tabela **active\_storage\_attachments** armazena informações sobre os anexos, como nome e tipo, enquanto a tabela **active\_storage\_blobs** mantém os dados binários reais dos arquivos. Essas tabelas são essenciais para o uso do **Active Storage**, uma funcionalidade do Rails para gerenciamento eficiente de uploads e downloads de arquivos, como imagens ou documentos. Já a **schema\_migrations** armazena as migrações para o banco.

## Tabelas Geradas para Usuário e Controle:

**Usuarios:** Tabela criada para armazenar e organizar dados dos clientes do projeto, são definidas como pessoas como pessoas únicas e sua tabela possui os seguintes atributos sendo eles id, email, encrypted\_password, prontuário, nome, tipo, telefone, data\_nascimento esses por sinal foram gerados pelas models criadas somente pelo programador.

- ID: Feito para organizar e singularizar o usuário como único, tipo BigInt.
- Email: Campo para armazenar o contato de email do usuário, tipo VarChar.
- Encrypted Password: Criado para armazenar as senhas encriptadas para maior segurança, tipo VarChar.
- Prontuário: Onde armazenamos o prontuário do usuário caso ele o possua, tipo VarChar.
- Nome: Forma padrão para nomear o usuário, tipo VarChar.
- CPF: Campo principal para identificar a pessoa que está cadastrada, tipo VarChar.
- Tipo: Sua principal função é dizer qual o cargo o usuário possui influenciando suas limitações no sistema, possuindo as seguintes opções aluno, admin, secretario, porteiro, visitante. Tipo Int (possuem o tipo Int por serem enumerados de 1 a 5 no sistema)
- Telefone: Aqui ficam disponíveis o telefone para contato do usuário, tipo VarChar.
- Data de Nascimento: Essencial para calcular a idade e saber se o usuário está autorizado a sair com o consentimento dos pais caso seja de menos, tipo DateTime.

**Demais Campos:** Os campos reset\_password\_token, jti, reset\_password\_sent\_at, remember\_created\_at, created\_at e updated\_at são criados pelo Rails para serem associados à funcionalidade de redefinição de senha em aplicações Ruby on Rails ou armazenar edições no banco.

- Reset Password Token: Armazena o token único gerado quando um usuário solicita a redefinição de senha, tipo VarChar.
- Reset Password sent: Registra o timestamp do momento em que o token de redefinição de senha foi enviado, tipo DateTime.
- Remember Created: Usado para rastrear quando um usuário opta por "lembrar-se" ao fazer login, permitindo sessões persistentes, tipo DateTime.
- Created\_at e Updated\_at: Indicam os timestamps de criação e última atualização de um registro no banco de dados, fornecendo informações sobre sua vida útil, tipo DateTime.
- JTI: Utilizado para identificar exclusivamente um token JWT (JSON Web Token). Este campo armazena um identificador único para cada token JWT emitido, tipo VarChar.

## GemFile

### Explicando a GemFile:

**O que é GemFile:** O Gemfile é um componente fundamental em projetos Ruby, especialmente naqueles construídos com o framework Ruby on Rails. Trata-se de um arquivo de configuração que descreve as dependências do projeto, especificando quais gems (bibliotecas ou pacotes Ruby) e suas versões são necessárias para que o projeto funcione corretamente.

O Gemfile é essencial para a padronização e reprodução de ambientes de desenvolvimento. Ele simplifica a instalação e atualização de dependências, tornando a gestão de bibliotecas Ruby eficiente e consistente em diferentes máquinas e ambientes de desenvolvimento. Ao compartilhar o Gemfile com outros membros da equipe, é possível garantir que todos estejam usando as mesmas versões das gems, minimizando problemas relacionados a inconsistências de dependências.

**Como Utilizar:** O Gemfile é utilizado em conjunto com o Bundler, uma ferramenta de gerenciamento de dependências para Ruby. O Bundler lê o Gemfile e instala as gems listadas, garantindo que todas as versões especificadas sejam atendidas, evitando conflitos entre diferentes versões de gems, por meio do comando **bundle install** o gerenciador começara a instalação das gems listadas.

### GemFile do Projeto:

1. **Ruby:** Source 'https://rubygems.org': Define a fonte padrão para as gems, indicando que as gems devem ser baixadas do repositório oficial de gems Ruby.
2. **Git\_source:** (:github) { |repo| "https://github.com/#{repo}.git" }: Especifica que as gems provenientes do GitHub devem ser obtidas a partir do repositório do GitHub correspondente.
3. **Ruby '3.1.2':** Define a versão mínima do Ruby necessária para o projeto.
4. **Gem:** Gem 'rails', '~> 6.1.6': Adiciona a gem do Rails ao projeto, especificando que deve ser utilizada a versão 6.1.6 ou superior, mas inferior à 6.2.
5. **MySQL2:** Gem 'mysql2', '~> 0.5': Adiciona a gem do MySQL2 como adaptador de banco de dados para o Active Record, especificando que deve ser utilizada a versão 0.5 ou superior.
6. **Puma:** Gem 'puma', '~> 5.0': Adiciona a gem do Puma como servidor de aplicação, especificando que deve ser utilizada a versão 5.0 ou superior.
7. **CORS:** Gem 'rack-cors': Adiciona a gem Rack CORS para lidar com Cross-Origin Resource Sharing (CORS), permitindo chamadas AJAX entre origens diferentes.

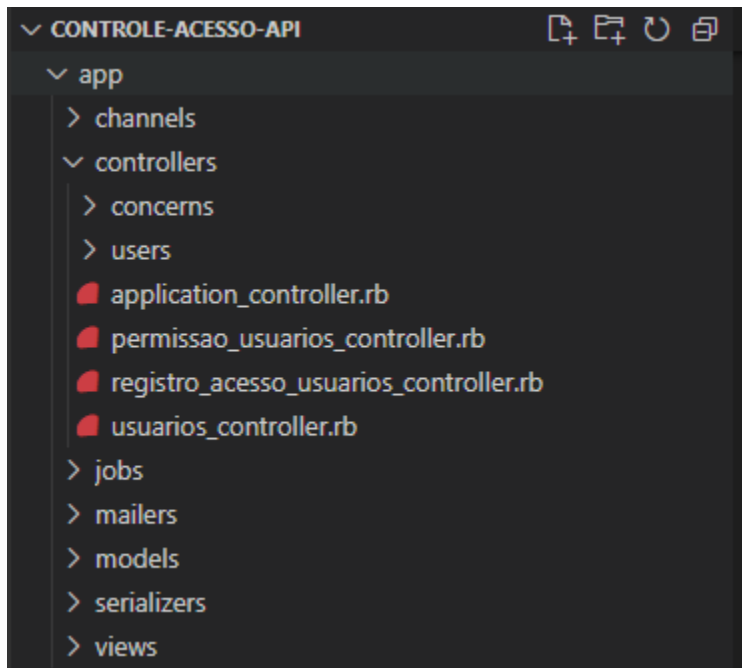


8. **Devise:** Devise é uma poderosa gem de autenticação para Ruby on Rails. Ela simplifica o processo de implementação de autenticação e autorização em um aplicativo Rails, ele fornece funcionalidades como registro de usuários, confirmação por e-mail, redefinição de senhas, controle de sessão, entre outros.
9. **Devise-JWT:** Devise-JWT é uma extensão para o Devise que adiciona suporte para autenticação baseada em JSON Web Tokens (JWT). Ademais ele permite que o Devise utilize tokens JWT para autenticação, o que é útil em aplicações modernas devido à sua simplicidade, escalabilidade e capacidade de serem utilizados em ambientes distribuídos.
10. **Jsonapi-Serializer:** Jsonapi-Serializer é uma gem que facilita a serialização de recursos para o formato JSON API, também ajuda a estruturar e formatar os dados da sua aplicação de acordo com as especificações do JSON API, um formato padronizado para APIs RESTful.
11. **Jbuilder:** É uma gem utilizada para construir JSON de uma maneira mais expressiva e estruturada. Facilita a geração de respostas JSON customizadas em controladores Rails, permitindo uma sintaxe mais legível e manutenível.
12. **Dotenv-Rails:** O Dotenv-Rails simplifica a gestão de variáveis de ambiente em um projeto Ruby on Rails. Permite carregar variáveis de ambiente definidas em arquivos env no ambiente de desenvolvimento e de teste. Isso é especialmente útil para configurar chaves secretas, senhas e outras configurações sensíveis que variam entre os ambientes.
13. **Group:** Gevelopment do end: Define um grupo de gems específicas para o ambiente de desenvolvimento, incluindo gems como 'listen' e 'spring' que aceleram o desenvolvimento.
14. **Tzinfo Data:** Gem 'tzinfo-data', platforms: [:mingw, :mswin, :x64\_mingw, :jruby]: Adiciona a gem tzinfo-data, que fornece dados de fuso horário para ambientes onde o Windows não inclui esses arquivos.

**Função das Gems escolhidas:** Essas gems, quando combinadas, fornecem um conjunto abrangente de funcionalidades para autenticação, serialização de dados e manipulação de variáveis de ambiente em um projeto Ruby on Rails, contribuindo para o desenvolvimento de aplicativos web robustos, seguros e escaláveis.

## Controles

### Organização Controllers:



**Sobre os Controllers:** No Ruby on Rails, a pasta `app/controllers` é um diretório fundamental que contém os controladores da sua aplicação. Os controladores desempenham um papel crucial no padrão arquitetônico MVC (Model-View-Controller) do Rails, ajudando a gerenciar o fluxo de dados entre o modelo (que representa os dados da aplicação), a visão (que exibe a interface do usuário) e as diversas interações do usuário.

A estrutura da pasta `app/controllers` é geralmente organizada de acordo com os recursos e entidades do seu aplicativo. Cada arquivo dentro dessa pasta é um controlador Ruby, cujo nome é geralmente no plural para refletir a natureza dos recursos que ele manipula.

1. **Naming Conventions:** Os nomes dos controladores seguem convenções de nomenclatura padrão no Rails. Por exemplo, um controlador para a entidade `User` seria nomeado `UserController`. Isso segue a convenção de pluralização para refletir a manipulação de vários usuários.
2. **Herança:** Os controladores geralmente herdam de `ApplicationController`. O `ApplicationController` é o controlador base para toda a aplicação e pode conter métodos e configurações globais.

3. **Ações e Rotas:** Métodos dentro de um controlador são chamados de "ações". Cada ação geralmente corresponde a uma rota específica da sua aplicação. Por exemplo, uma ação index pode lidar com a exibição de todos os registros de um modelo.
4. **Respostas HTTP:** As ações nos controladores geralmente manipulam solicitações HTTP e retornam respostas. Isso pode incluir redirecionamentos, renderizações de visualizações ou até mesmo respostas JSON para APIs.
5. **Interação com Modelos:** Os controladores interagem com os modelos para buscar, criar, atualizar e excluir dados no banco de dados. Eles agem como intermediários entre as solicitações HTTP e a lógica de negócios encapsulada nos modelos.
6. **Visualizações:** As ações frequentemente renderizam visualizações, que estão localizadas na pasta app/views. As visualizações são responsáveis pela apresentação dos dados ao usuário.
7. **Filtros e Callbacks:** Os controladores podem incluir filtros e callbacks que são executados antes, durante ou após uma ação. Isso é útil para a execução de lógica comum a várias ações.
8. **RESTful Routes:** Os controladores no Rails seguem o padrão RESTful, o que significa que as ações geralmente correspondem às operações CRUD (Create, Read, Update, Delete) e são mapeadas para rotas RESTful.

**Application Controller:** Define um controlador base chamado ApplicationController que herda de ActionController::API. Três métodos (checar\_admin, checar\_secretario, checar\_porteiro) são implementados para verificar as permissões do usuário atual em relação a diferentes papéis (admin, secretário, porteiro). Se o usuário não tiver as permissões adequadas, uma resposta JSON de erro é renderizada com status não autorizado. Este código é usado para controlar o acesso a recursos com base nos papéis dos usuários no sistema.

```
app > controllers > application_controller.rb
1 class ApplicationController < ActionController::API
2   def checar_admin
3     if current_usuario.admin?
4       return true
5     else
6       render json: { error: 'Não tem permissão para acessar os usuarios do sistema!' }, status: :unauthorized
7     end
8   end
9
10  def checar_secretario
11    if current_usuario.admin? || current_usuario.secretario?
12      return true
13    else
14      render json: { error: 'Não tem permissão para acessar os usuarios do sistema!' }, status: :unauthorized
15    end
16  end
17
18  def checar_porteiro
19    if current_usuario.admin? || current_usuario.porteiro?
20      return true
21    else
22      render json: { error: 'Não tem permissão para acessar os usuarios do sistema!' }, status: :unauthorized
23    end
24  end
25 end
26
```

[Link\(https://github.com/JacksonGregorio/Controle-Acesso-IFSP-API/blob/main/app/controllers/application\\_controller.rb\)](https://github.com/JacksonGregorio/Controle-Acesso-IFSP-API/blob/main/app/controllers/application_controller.rb)

**Permissao Usuarios Controller:** Possui o controlador PermissaoUsuariosController herda do ApplicationController e implementa ações para gerenciar permissões de usuários. Antes de executar as ações, ele exige autenticação de usuário e verifica se o usuário tem a função de secretário. As ações incluem listar permissões existentes, criar uma nova permissão para um usuário específico (com validações) e definir um conjunto de antes da ação para configurar o usuário alvo. Este código é responsável por controlar o acesso e manipulação de permissões de usuários, garantindo restrições apropriadas com base nas funções e características do usuário.

```
1 class PermissaoUsuariosController < ApplicationController
2   before_action :authenticate_usuario!
3   before_action :checar_secretario
4
5   before_action :setar_usuario
6   before_action :checar_aluno, only: [:criar_permissao]
7
8   def listar_permissoes
9     @permissao_usuarios = @usuario.permissao_usuarios
10    render :index
11  end
12
13  def criar_permissao
14    @permissao_usuario = @usuario.permissao_usuarios.new(permissao_usuario_params)
15
16    if @permissao_usuario.save
17      render :show, status: :created
18    else
19      render json: @permissao_usuario.errors, status: :unprocessable_entity
20    end
21  end
22
23  private
24
25  def checar_aluno
26    if @usuario.tipo != 'aluno'
27      render json: { error: 'Usuário não é um aluno' }, status: :unprocessable_entity
28      return
29    end
30
31    if @usuario.adulto?
32      render json: { error: 'Usuário é maior de idade' }, status: :unprocessable_entity
33      return
34    end
35  end
36
37  def setar_usuario
38    @usuario = Usuario.find(params[:id])
39  end
40
41  def permissao_usuario_params
42    params.permit(:data_inicio, :data_fim, :descricao_permissao)
43  end
44 end
45
```

*Link([https://github.com/JacksonGregorio/Controle-Acesso-IFSP-API/blob/main/app/controllers/permissao\\_usuarios\\_controller.rb](https://github.com/JacksonGregorio/Controle-Acesso-IFSP-API/blob/main/app/controllers/permissao_usuarios_controller.rb))*

**Registro Acesso Usuarios Controller:** O controlador RegistroAcessoUsuariosController gerencia o registro de acessos de usuários, exigindo autenticação prévia. A ação registrar\_acesso permite que um porteiro registre o acesso de um usuário, armazenando o tipo de acesso. A ação listar\_registros é acessível apenas para administradores e retorna registros de acesso filtrados por data, prontuário e CPF do usuário. O código também inclui métodos privados para filtrar registros por data, prontuário e CPF, além de definir o usuário alvo com base em prontuário ou CPF.

```
1 class RegistroAcessoUsuariosController < ApplicationController
2   before_action :authenticate_usuario!
3
4   before_action :checar_admin, only: [:listar_registros]
5   before_action :checar_porteiro, only: [:registrar_acesso]
6
7   before_action :setar_usuario, only: [:registrar_acesso]
8
9   def registrar_acesso
10    @registro_acesso_usuario = @usuario.registro_acesso_usuarios.new(registrar_acesso_params)
11
12    if @registro_acesso_usuario.save
13      render :show, status: :created
14    else
15      render json: @registro_acesso_usuario.errors, status: :unprocessable_entity
16    end
17  end
18
19  def listar_registros
20    @registro_acesso_usuarios = RegistroAcessoUsuario.all.order(created_at: :desc)
21
22    filtrar_por_data if params[:created_at].present?
23    filtrar_por_prontuario if params[:prontuario].present?
24    filtrar_por_cpf if params[:cpf].present?
25  end
26  render :index
27 end
28
29 private
30
31 def registrar_acesso_params
32   params.permit(:tipo)
33 end
34
35 def setar_usuario
36   begin
37     if params[:prontuario].present?
38       @usuario = Usuario.find_by(prontuario: params[:prontuario])
39     elsif params[:cpf].present?
40       @usuario = Usuario.find_by(cpf: params[:cpf])
41     else
42       render json: { error: "Prontuário ou CPF não informado" }, status: :bad_request
43     end
44   rescue ActiveRecord::RecordNotFound => e
45     render json: { error: e.message }, status: :not_found
46   end
47 end
48
49 def filtrar_por_data
50   data = Date.parse(params[:created_at]).to_datetime
51   @registro_acesso_usuarios = @registro_acesso_usuarios.where(created_at: data.beginning_of_day..data.end_of_day)
52 end
53
54 def filtrar_por_prontuario
55   @registro_acesso_usuarios = @registro_acesso_usuarios.joins(:usuario).where("usuarios.prontuario LIKE ?", "%#{params[:prontuario]}%")
56 end
57
58 def filtrar_por_cpf
59   @registro_acesso_usuarios = @registro_acesso_usuarios.joins(:usuario).where("usuarios.cpf LIKE ?", "%#{params[:cpf]}%")
60 end
61 end
62
```

Link([https://github.com/JacksonGregorio/Controle-Acesso-IFSP-API/blob/main/app/controllers/registro\\_acesso\\_usuarios\\_controller.rb](https://github.com/JacksonGregorio/Controle-Acesso-IFSP-API/blob/main/app/controllers/registro_acesso_usuarios_controller.rb))

**Usuarios Controller:** O controlador UsuariosController no Ruby on Rails gerencia operações relacionadas aos usuários do sistema. Antes de executar as ações, o código requer autenticação do usuário e verifica diferentes níveis de permissão (admin, secretário, porteiro). As ações incluem a criação de visitantes com senhas geradas automaticamente, listagem de todos os usuários, listagem específica de alunos (com opção de filtrar por prontuário), e atualização de informações do usuário. O código utiliza métodos privados para filtrar usuários por prontuário, configurar o usuário alvo e gerenciar parâmetros permitidos.

```
1 class UsuariosController < ApplicationController
2   before_action :authenticate_usuario!
3
4   before_action :checar_admin, only: [:atualizar_usuario, :listar_usuarios]
5   before_action :checar_secretario, only: [:listar_alunos]
6   before_action :checar_porteiro, only: [:criar_visitante]
7
8   before_action :setar_usuario, only: [:atualizar_usuario]
9
10  def criar_visitante
11    @usuario = Usuario.new(usuario_params)
12    @usuario.tipo = :visitante
13    @usuario.password = Devise.friendly_token.first(8)
14    @usuario.registro_acesso_usuarios.new(tipo: :entrada)
15
16    if @usuario.save
17      render :show, status: :created
18    else
19      render json: @usuario.errors, status: :unprocessable_entity
20    end
21  end
22
23  def listar_usuarios
24    @usuarios = Usuario.all
25    render :index
26  end
27
28  def listar_alunos
29    @usuarios = Usuario.where(tipo: 'aluno')
30    filtrar_por_prontuario if params[:prontuario].present?
31
32    render :index
33  end
34
35  def atualizar_usuario
36    if @usuario.update(params_usuario)
37      render :show
38    else
39      render json: { error: 'Não foi possível atualizar o usuário!' }, status: :unprocessable_entity
40    end
41  end
42
43  private
44
45  def usuario_params
46    params.permit(:email, :nome, :cpf, :telefone, :foto)
47  end
48
49  def filtrar_por_prontuario
50    @usuarios = @usuarios.where("prontuario LIKE ?", "%#{params[:prontuario]}%")
51  end
52
53  def setar_usuario
54    @usuario = Usuario.find(params[:id])
55  end
56
57  def params_usuario
58    params.permit(:tipo)
59  end
60 end
61
```

Link([https://github.com/JacksonGregorio/Controle-Acesso-IFSP-API/blob/main/app/controllers/usuarios\\_controller.rb](https://github.com/JacksonGregorio/Controle-Acesso-IFSP-API/blob/main/app/controllers/usuarios_controller.rb))

## Sobre o Users:

**Funcionalidade do Users:** Em resumo, esse controlador personalizado gerencia as operações de login e logout para usuários, fornecendo respostas em formato JSON e incluindo verificações de autenticação através do token JWT.

## Rotas

Sobre as Rotas:

```
1 Rails.application.routes.draw do
2   devise_for :usuarios, path: '', path_names: {
3     sign_in: 'login',
4     sign_out: 'logout',
5     registration: 'signup'
6   },
7   controllers: {
8     sessions: 'users/sessions',
9     registrations: 'users/registrations'
10  }
11
12  # usuarios
13  post '/visitantes', to: 'usuarios#criar_visitante'
14
15  get '/usuarios', to: 'usuarios#listar_usuarios'
16  put '/usuarios/:id', to: 'usuarios#atualizar_usuario'
17
18  get '/alunos', to: 'usuarios#listar_alunos'
19
20  # registro_acesso_usuarios
21  post '/registro_acesso_usuarios', to: 'registro_acesso_usuarios#registrar_acesso'
22  get '/registro_acesso_usuarios', to: 'registro_acesso_usuarios#listar_registros'
23
24  # permissao_usuarios
25  post '/usuarios/:id/permissoes_usuario', to: 'permissao_usuarios#criar_permissao'
26  get '/usuarios/:id/permissoes_usuario', to: 'permissao_usuarios#listar_permissoes'
27 end
28
```



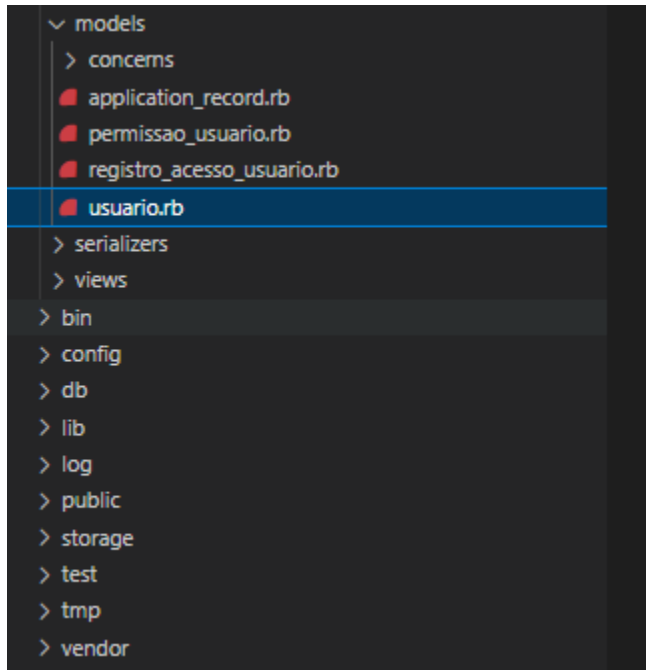
Link(<https://github.com/JacksonGregorio/Controle-Acesso-IFSP-API/blob/main/config/routes.rb>)

**Rota Navegador:** Essa rota tem a função de conectar o usuário a aplicação por meio de ferramentas que dá suporte à documentação das requisições ou uma API de sincronização. Ela também tem a função de passar parâmetros para aplicação nesse caso por meio do `/:id` que passa os dados para o Backend.

**Rota Função:** Basicamente rastreia a função que será usado naquela rota de navegação, através do caminho `"usuarios#"` que localiza a file que está o código e `"atualizar_usuario"` que rastreia a função que será utilizada da file.

## Models

### Organização das Models:



**Sobre as Models:** Em Ruby on Rails, models são classes que representam e interagem com os dados do banco de dados. Elas desempenham um papel central no padrão de arquitetura MVC (Model-View-Controller) do Rails. Uma model representa uma tabela no banco de dados e é responsável por definir como os dados são estruturados e manipulados. Cada instância da model representa uma entrada específica na tabela, models interagem com o banco de dados para realizar operações CRUD (Create, Read, Update, Delete). Elas fornecem uma interface de programação para recuperar, salvar, atualizar e excluir registros no banco de dados. Models incluem lógica para validar os dados antes de serem salvos no banco de dados isso garante que os dados cumpram certos critérios, como presença, formato ou unicidade.



```

1 class Usuario < ApplicationRecord
2   include Devise::JWT::RevocationStrategies::JTIMatcher
3   # Include default devise modules. Others available are:
4   # :confirmable, :lockable, :timeoutable, :trackable and :omniauthable
5   devise :database_authenticatable, :registerable,
6         :recoverable, :rememberable, :validatable, :jwt_authenticatable, jwt_revocation_strategy: self,
7         authentication_keys: [:prontuario]
8
9   has_many :permissao_usuarios
10  has_many :registro_acesso_usuarios
11
12  has_one_attached :foto
13
14  enum tipo: [:outros_colaboradores_campus, :aluno, :admin, :secretario, :porteiro, :visitante]
15
16  def admin?
17    tipo == 'admin'
18  end
19
20  def secretario?
21    tipo == 'secretario'
22  end
23
24  def porteiro?
25    tipo == 'porteiro'
26  end
27
28  def adulto?
29    Time.now.to_date >= (data_nascimento.to_date + 18.years)
30  end
31
32  def autorizado?
33    if tipo == 'aluno'
34      adulto? || menor_idade_autorizado?
35    else
36      true
37    end
38  end
39
40  private
41
42  def menor_idade_autorizado?
43    permissao_usuarios.where('data_inicio <= ? AND data_fim >= ?', Time.now, Time.now).any?
44  end
45 end
46

```

*Link(<https://github.com/JacksonGregorio/Controle-Acesso-IFSP-API/blob/main/app/models/usuario.rb>)*

**Usuario:** O modelo Usuario incorpora funcionalidades de autenticação Devise, possui associações com outras tabelas, enumera tipos de usuários, e fornece métodos para verificar o tipo, idade e autorização do usuário. Essas funcionalidades são essenciais para a gestão de usuários em um aplicativo Rails.

```

1  class RegistroAcessoUsuario < ApplicationRecord
2    belongs_to :usuario
3
4    enum tipo: [:entrada, :saida]
5    validate :checar_usuario_permissao
6
7    def checar_usuario_permissao
8      if !self.usuario.autorizado? && self.tipo == 'saida'
9        errors.add(:usuario, 'Usuário não está autorizado')
10     end
11   end
12 end
13

```

Link([https://github.com/JacksonGregorio/Controle-Acesso-IFSP-API/blob/main/app/models/registro\\_acesso\\_usuario.rb](https://github.com/JacksonGregorio/Controle-Acesso-IFSP-API/blob/main/app/models/registro_acesso_usuario.rb))

**Registro Acesso Usuário:** Este código define o modelo RegistroAcessoUsuario em um aplicativo Ruby on Rails. Ele está associado a um usuário e possui uma enumeração de tipos para representar entrada ou saída. Além disso, inclui uma validação personalizada, `checar_usuario_permissao`, que verifica se o usuário associado tem permissão para sair, exibindo um erro se não estiver autorizado. Esse modelo é utilizado para registrar acessos de usuários, assegurando que as saídas estejam de acordo com as permissões definidas para cada usuário.

**Outras Models:** Os outros modelos colocam valores e atributos aos usuários, `permissao_usuario` seta o valor da foto e imagem e `application_record` configura o abstract da classe, ou seja eles basicamente acertam os outros modelos.

■ ■ ■