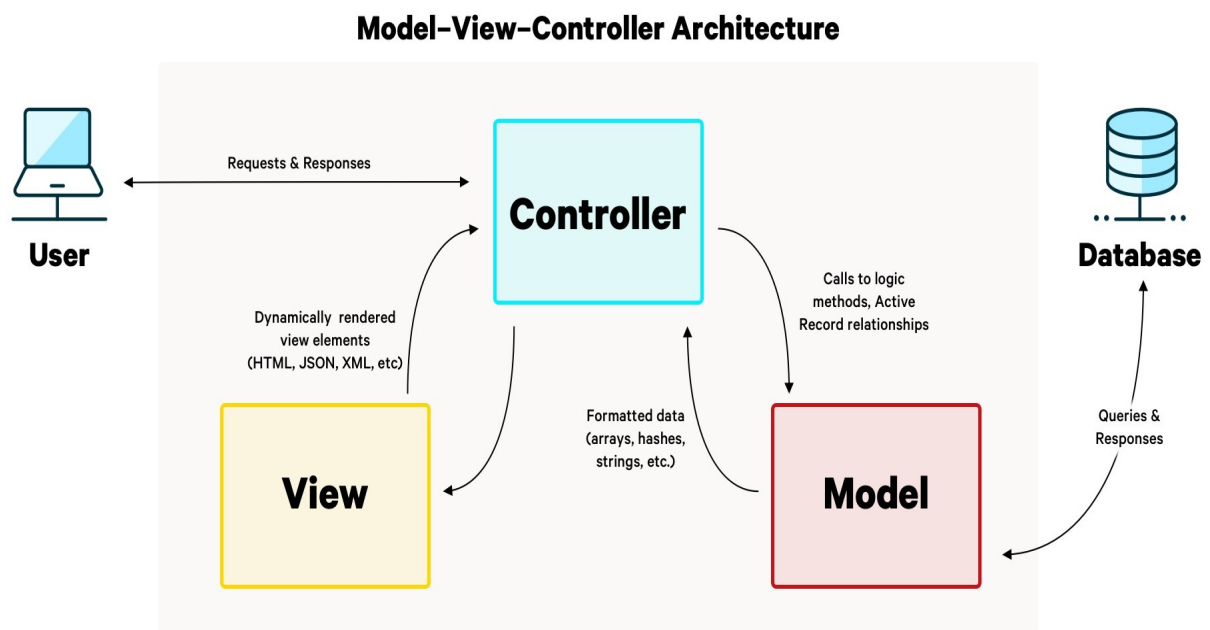


1-MVC

O código Rails é organizado usando a arquitetura Model-View-Controller (MVC). Com MVC, temos três conceitos principais onde a maioria do nosso código vive:

- Modelo - Gerencia os dados em seu aplicativo. Normalmente, suas tabelas de banco de dados.
- Exibir - Lida com respostas de renderização em diferentes formatos, como HTML, JSON, XML, etc.
- Controlador - Lida com as interações do usuário e a lógica de cada solicitação.



Resumo do Fluxo Completo

- 1 **Usuário acessa a URL** (GET /users/1).
- 2 **Router direciona a requisição** para UsersController#show.
- 3 **Controller processa a requisição** e chama o Model (User.find).
- 4 **Model busca os dados no banco** e retorna um objeto User.
- 5 **Controller passa os dados para a View**.
- 6 **View renderiza o HTML com os dados do usuário**.
- 7 **Rails envia a resposta ao navegador** e o usuário vê a página.

Agora que temos uma compreensão básica do MVC, vamos ver como ele é usado no Rails.

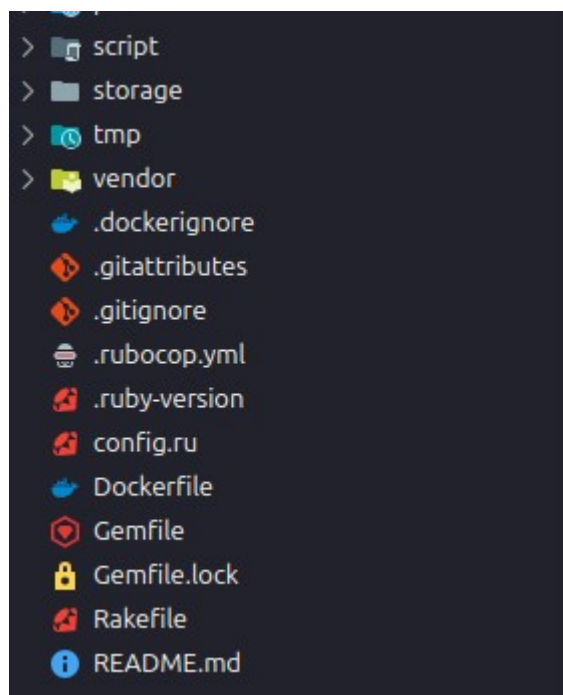
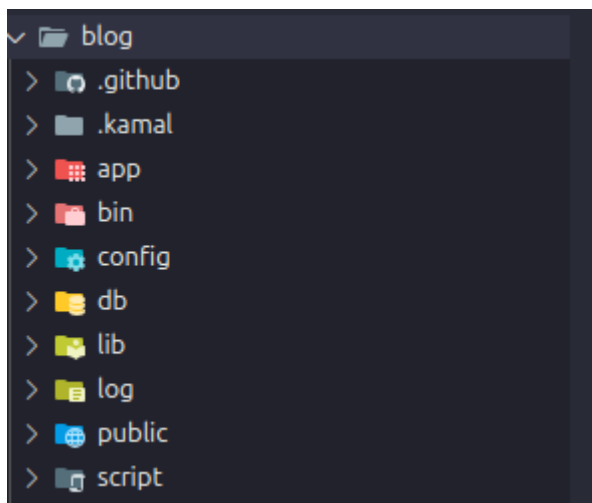
2-CRIAÇÃO DE PROJETOS

a) rails new blog -T --database=postgresql

-1. Explicação dos Componentes do Comando

- **rails new blog** → Cria um novo projeto Rails chamado "**blog**" dentro de um diretório com esse nome.
- **-T** → Exclui os arquivos e configurações padrão para testes (`Test::Unit`). Se quiser usar **RSpec** ou outro framework de testes, esse flag é útil.
- **--database=postgresql** → Configura o projeto para usar o **PostgreSQL** como banco de dados em vez do padrão (**SQLite**).

b) Após a criação ele inicia com uma estrutura de pastas



c) o que é cada pasta

Sim, ao rodar o comando `rails new blog -T --database=postgresql`, o Rails cria uma estrutura de diretórios e arquivos padrão para o novo projeto. Essa estrutura organiza o código e os recursos necessários para uma aplicação Rails funcionar.

Aqui estão as **pastas principais** que o Rails cria para você:

1. app/

Contém o código principal da sua aplicação:

- **models/**: Contém os modelos (como `User`, `Post`, etc.) que representam os dados e interagem com o banco de dados.
- **controllers/**: Contém os controladores que gerenciam a lógica de negócios e interagem com as views e modelos.
- **views/**: Contém as views que são responsáveis pela interface do usuário (arquivos `.html.erb`).

- **helpers/**: Contém módulos para ajudar na geração de conteúdo nas views (funções reutilizáveis).
- **assets/**: Contém arquivos estáticos como CSS, JavaScript e imagens.
- **mailers/**: Contém classes responsáveis por enviar e-mails (caso tenha implementado esse recurso).

2. config/

Contém arquivos de configuração para diferentes aspectos da aplicação:

- **database.yml**: Configurações do banco de dados (definido para PostgreSQL com o comando que você executou).
- **routes.rb**: Define as rotas da aplicação, que associam URLs a controladores e ações.
- **environments/**: Contém configurações específicas para os ambientes de desenvolvimento, produção e teste.

3. db/

Contém arquivos relacionados ao banco de dados:

- **migrate/**: Armazena as migrations, que são arquivos Ruby responsáveis por criar ou modificar as tabelas no banco de dados.
- **seeds.rb**: Usado para popular o banco de dados com dados iniciais (dados fictícios, por exemplo).

4. lib/

Contém código que não se encaixa diretamente nas pastas **app**, como bibliotecas externas, módulos ou classes auxiliares.

5. public/

Contém arquivos estáticos acessíveis diretamente pela web, como imagens, JavaScript e arquivos CSS.

6. test/ (ou spec/ se você usar RSpec)

Contém os arquivos de teste (como unitários e de integração). Como você usou o flag **-T**, essa pasta será criada sem arquivos de testes, pois você provavelmente deseja usar outra ferramenta de testes (como o RSpec).

7. tmp/

Usado para armazenar arquivos temporários, como cache e sessões.


8. vendor/

Contém dependências externas ou gems que não são gerenciadas pelo Bundler. Normalmente, você não vai mexer aqui diretamente.

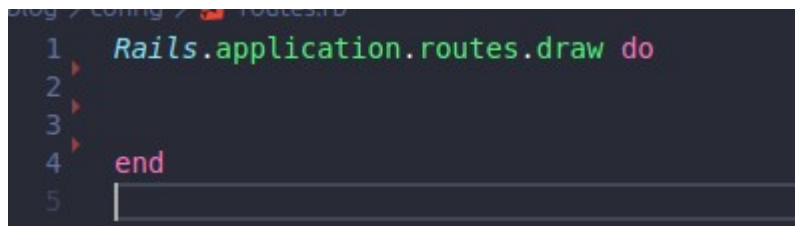
3-iniciando um projeto

-Sempre que iniciar um projeto rails ou qualquer resposta a tasks no projeto deverá se iniciar com uma ROTA, exemplo de task “criar uma página de bem vindo”.

Projeto → app → config → routes.rb



-Ao acessar terá acesso ao bloco de definição de rotas onde iniciaremos uma nova rota.



-Nesse bloco de definição você deve especificar as requisições, especificamente esse bloco tratará de requisições HTTP (GET, POST, etc) , assim quando escolher o método de requisição você estará definindo quais controllers e ações serão invocadas.

1. GET

- Descrição: Usado para recuperar informações do servidor.
- Exemplo: Quando você acessa uma página em um navegador, uma requisição GET é feita.
- Uso típico: Buscar dados, como exibir uma página, uma lista de posts, etc.

2. POST

- Descrição: Usado para enviar dados ao servidor, geralmente para criar um novo recurso.
- Exemplo: Enviar um formulário para criar um novo post em um blog.
- Uso típico: Criar novos dados no servidor.

3. PUT

- Descrição: Usado para atualizar um recurso existente no servidor, substituindo-o por um novo.
- Exemplo: Atualizar um post existente com novos dados.
- Uso típico: Atualizar todos os dados de um recurso.
- Exemplo de rota em Rails:

4. PATCH

- Descrição: Semelhante ao PUT, mas usado para atualizar parcialmente um recurso.
- Exemplo: Atualizar um único campo de um post, como alterar apenas o título ou o conteúdo.
- Uso típico: Atualizar parcialmente os dados de um recurso.

5. DELETE

- Descrição: Usado para excluir um recurso do servidor.
- Exemplo: Deletar um post de um blog.
- Uso típico: Excluir dados no servidor.

6. HEAD

- Descrição: Semelhante ao GET, mas apenas para obter os cabeçalhos da resposta sem o corpo (útil para verificar se o recurso existe).
- Exemplo: Verificar se uma página está disponível, sem baixar o conteúdo.
- Uso típico: Obter apenas os cabeçalhos de uma resposta, sem o corpo.

Assim vamos fazer uma requisição quando usuário for acessar a página bem vindo

```
Rails.application.routes.draw do
  get '/', Controller:'home', Action:'index'
end
```

Esse código está definindo uma rota para a URL raiz (/) do seu site. Quando alguém acessar essa URL, o Rails vai chamar o **método index** do **HomeController** para lidar com a requisição e gerar a resposta ao usuário.

-Descobrir se a rota está criada

O comando `rails routes -g home` é usado no Ruby on Rails para listar todas as rotas definidas no seu aplicativo que correspondem ao termo "home". A opção `-g` faz com que o Rails filtre as rotas e mostre apenas aquelas que contêm "home" no nome, controlador ou ação, abre o terminal do projeto e ele devolverá a resposta.

```
• → blog git:(main) x rails routes -g home
Prefix Verb URI Pattern Controller#Action
GET / home#index
```

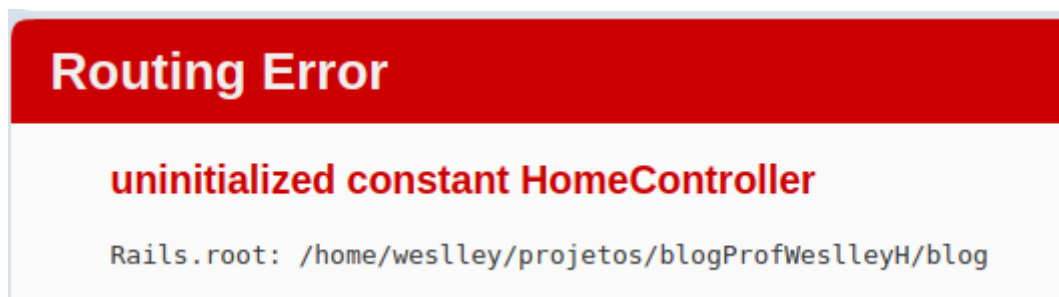
-Executando o projeto para testar a rota

No terminal da aplicação chamar a aplicação iniciamos ela com o comando `rails server` ou `s`, assim ela vai entregar a porta para acessarmos

```
GET / home#index
→ blog git:(main) x rails s
=> Booting Puma
=> Rails 8.0.2 application starting in development
```

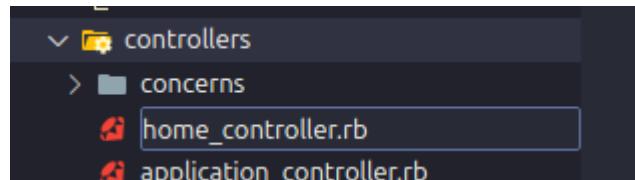
```
* PID: 21710
* Listening on http://127.0.0.1:3000
* Listening on http://[::]:3000
Use Ctrl-C to stop
```

Em seguida acessamos e deverá mostrar um erro



*****DEVEMOS APRENDER A LER OS ERROS E ENTENDER O QUE ELES ESTÃO APONTANDO*****

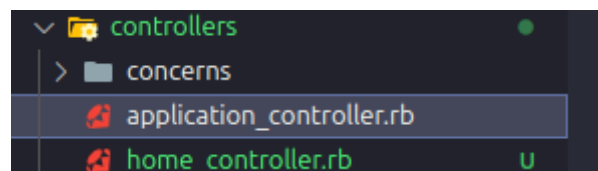
Nesse caso eles está apontando que não achou o controller, desse modo criaremos o controle com o nome que ele apontou, assim em app → controller , crie um arquivo com o nome.



Dentro do controller será criado a class que seguirar o formato que o erro avisou, nesse instante ela se comporta como uma classe comum de ruby.

```
blog > app > controllers > home_controller.rb
1  class HomeController
2
3  end
```

Além disso, observamos a existencia de um controller originario da criação do projeto que tem todos poderes no rails (**application_controller.rb**)



O **application_controller.rb** é um arquivo central no framework Ruby on Rails. Ele define o controlador base que outros controladores no seu aplicativo herdam. Ou seja, todas as classes de controladores no Rails herdam do ApplicationController, a menos que você defina explicitamente uma herança diferente, assim falaremos para o nosso HomeController herde.

```
blog > app > controllers > home_controller.rb
1  class HomeController < ApplicationController
2
3  end
```

observe que o modo de atualização da pagina é em tempo real, assim as modificações são salvas e executadas, assim ao aprovamos a herança do controller ele já aponta para outra tarefa.

Unknown action

The action 'index' could not be found for HomeController

Dessa vez, ele está apontando que está faltando uma função dentro do nosso controller, função index que apontamos no momento da criação da rota.

No view template for interactive request

HomeController#index is missing a template for request formats: text/html

Agora a nossa rota está direcionada de forma correta, tendo os passos de sua criação e action executados, assim vamos devolver uma view para o usuário, na pasta view o nome do controller será o nome da pasta e o nome da action da o nome para pagina e assim criasse com .html e erb para usar html e ruby

