Wednesday - 8 August, 2018

(http://blog.desafiolatam.com)

(http://desafiolatam.com)

TUTORIALES

MOTIVACIÓN (HTTP://BLOG.DESAFIOLATAM.COM/CATEGORY/MOTIVACION/)

GRADUADOS (HTTP://BLOG.DESAFIOLATAM.COM/CATEGORY/GRADUADOS/)

VER CURSOS (HTTP://DESAFIOLATAM.COM/)

Desafio Latam (Http://Blog.Desafiolatam.Com) > Tutoriales (Http://Blog.Desafiolatam.Com/Category/Tutoriales/) > Rails (Http://Blog.Desafiolatam.Com/Category/Tutoriales/Rails/) > APIs (Http://Blog.Desafiolatam.Com/Category/Tutoriales/Rails/Apis/) > Creando Servicios Web En Rails

CREANDO SERVICIOS WEB EN RAILS

¿QUÉ SON LOS SERVICIOS WEB?

Los servicios web son sistemas de software diseñados para comunicación máquina a máquina sobre una red (principalmente Internet). **En palabras sencillas son como páginas web pero para ser accesadas por programas en lugar de seres humanos**.

Estos servicios suelen ser APIs (Application Program interface) que pueden ser accedidos por máquinas que se encuentren conectadas a la misma red (clientes web) y son ejecutados dentro del sistema que los aloja (el servidor).

Servicios Web Estilo REST

REST es un estilo de arquitectura de software para diseños hipermedia distribuidos que define una colección de principios que resumen cómo los recursos del servidor web que aloja los servicios son accedidos, definidos y divididos. Generalmente cualquier interfaz (API) que transmite datos específicos de un dominio sobre HTTP sin una capa adicional es descrito como API REST.

Este estilo de arquitectura denominado REST está basado en estándares:

- HTTP
- URL/URI
- Representación de recursos: XML/HTML/GIF/JPEG/JSON...
- Tipos MIME: text/xml, text/html...

La WEB es REST

Muchos de los principios de La Web son la base de rest. Por ejemplo los estandarés que definen la web son: el protocolo HTTP, tipos de contenido: HTTP/GIF/JPG/...,
y otras tecnologías tales como el DNS.

HTTP es un protocolo clave en las arquitecturas REST. Posee una interfaz uniforme para acceder a los recursos, el cual consiste en URIs, métodos, códigos
de estado, cabeceras y un contenido guiado por tipos MIME.

Los métodos HTTP más importantes son PUT, GET, POST y DELETE... Les parece conocido?

Rails está basado en REST

El termino REST fue introducido en el año 2000 por Roy Fielding y desde entonces se empezó a popularizar dentro de los diseños de software de esa época y al día de hoy es de los más utilizados junto con SOAP.

Por esta razón muchos frameworks como rails usan este estilo para el desarrollo de aplicaciones. Es por eso que rails nos abstrae los principios de
este estilo de arquitectura.

Sin embargo rails es un framework para desarrollar aplicaciones web que van a ser intereactuadas principalmente por humanos y no por máquinas. Pero esto no quiere decir que con rails no se puedan hacer servicios web, al contrario, es mucho más sencillo que hacer aplicaciones web.

Manos a la obra

Scaffold normal

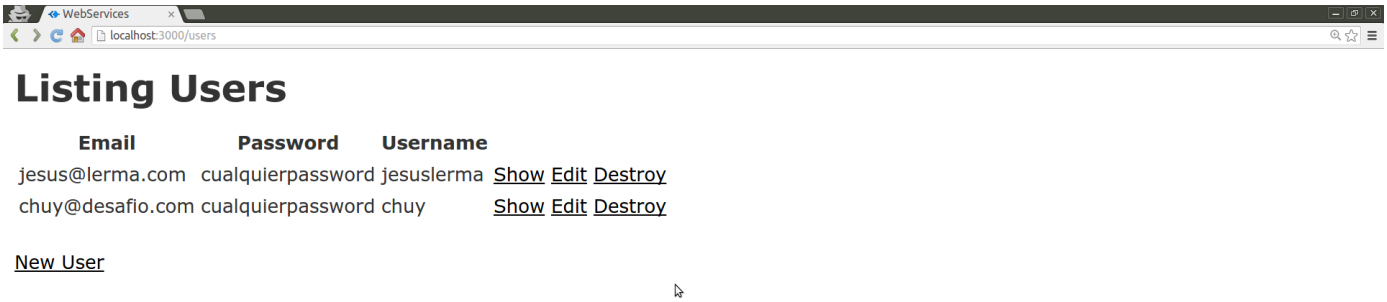
Empezamos creando un proyecto desde cero con el recurso que cualquier proyecto necesita: users.

```ruby
$ rails new web_services -d postgresql
$ cd web_services
$ git init
$ git add .
$ git commit -m "basic project structure"
$ rails g scaffold user email password username
$ rake db:create
$ rake db:migrate
$ git add .
$ git commit -m "add users scaffold"
```

Por último configuramos index como vista parcial

```ruby
# en el archivo config/routes.rb
root 'users#index'
```

Al entrar a localhost veremos el la vista de users. Agregamos 2 users



(http://blog.desafiolatam.com/wp-content/uploads/2016/05/users_index.png)

Creando nuestra API de users

El siguiente paso es empezar a crear un servicio web para users. Para esto es importante crear una carpeta llamada 'api' dentro de 'controllers' con
la finalidad de tener una mejor estructura de nuestro proyecto.

```ruby
$ rails g controller api/users --skip-template-engine --skip-helper --skip-assets
```

Como mencionamos al inicio, para los servicios web no es necesario crear una vista para el usuario final. Lo que nos interesa es que otra computadora pueda interpretar nuestros servicios, por lo tanto eliminamos las vistas `--skip-template-engine`, los assets ` `--skip-assets`, y los helpers `--skip-helper`.

Si escribimos el comando `$ rake routes` desde línea de comandos nos daremos cuenta de que aún no están disponibles nuestros recursos. Por lo tanto hay que escribirlos manualmente en el archivo `config/routes.rb`

```ruby
namespace :api do
  resources :users
end
```

Si vemos los recursos disponibles con el comando `$ rake routes`, el recurso api/users ya estará disponible:

```
hyde :: web_services master $ rake routes
       Prefix Verb   URI Pattern                    Controller#Action
        users GET    /users(.:format)               users#index
              POST   /users(.:format)               users#create
     new_user GET    /users/new(.:format)           users#new
    edit_user GET    /users/:id/edit(.:format)      users#edit
         user GET    /users/:id(.:format)           users#show
              PATCH  /users/:id(.:format)           users#update
              PUT    /users/:id(.:format)           users#update
              DELETE /users/:id(.:format)           users#destroy
         root GET    /                              users#index
    api_users GET    /api/users(.:format)           api/users#index
              POST   /api/users(.:format)           api/users#create
 new_api_user GET    /api/users/new(.:format)       api/users#new
edit_api_user GET    /api/users/:id/edit(.:format)  api/users#edit
     api_user GET    /api/users/:id(.:format)       api/users#show
              PATCH  /api/users/:id(.:format)       api/users#update
              PUT    /api/users/:id(.:format)       api/users#update
              DELETE /api/users/:id(.:format)       api/users#destroy
```

(http://blog.desafiolatam.com/wp-content/uploads/2016/05/rutas_api.jpg)

Pero existen recursos que no nos interesan, ya que estos recursos se usan en general para ser vistas de usuario:

1. new_api_user GET /api/users/new(.:format) api/users#new
2. edit_api_user GET /api/users/:id/edit(.:format) api/users#edit

Así que vamos a poner solamente los recursos que nos interesan. Para esto en el archivo `config/routes.rb` debemos de poner lo siguiente:

```ruby
  resources :users, only: [:index, :create, :destroy, :update, :show]
```

Con esto ya tenemos solamente los recursos que estaremos usando. Sin embargo al acceder al recurso http://localhost:3000/api/users (http://localhost:3000/api/users) notaremos que nos marca
error: The action 'index' could not be found for Api::UsersController. Si vamos a nuestro archivo `controllers/api/users_controller.rb` notaremos que se encuentra vacio. Como usamos el generador de controller y no el de scaffold entonces se genera el controller completamente vacio. Tenemos que deja lo siguiente:

```ruby
class Api::UsersController < ApplicationController
  before_action :set_user, only: [:show, :update, :destroy]

  def index
    @users = User.all
    render json: @users
  end

  def show
  end

  def create
    @user = User.new(user_params)

    if @user.save
      render json: @user, status: :created
    else
      render json: @user.errors, status: :unprocessable_entity
    end
  end

  def update
    if @user.update(user_params)
      render json: @user, status: :ok
    else
      render json: @user.errors, status: :unprocessable_entity
    end
  end

  def destroy
    @user.destroy
    head :no_content
  end

  private
    def set_user
      @user = User.find(params[:id])
    end

    def user_params
      params.require(:user).permit(:email, :password, :username)
```

```
    end
end
```

Si accedemos nuevamente al recurso pero pidiendo el formato en json:
http://localhost:3000/api/users.json (http://localhost:3000/api/users.json). Podremos ver una lista de usuarios pero
esta vez en formato json:

Ruby

```
[{
  "id":1,
  "email":"jesus@lerma.com",
  "password":"cualquierpassword",
  "username":"jesuslerma",
  "created_at":"2016-05-16T02:26:47.038Z",
  "updated_at":"2016-05-16T02:26:47.038Z"
}...]
```

Esto es debido a que estamos especificando que el formato será json al acceder al recurso users.json. Para evitar tener que poner esto cada que queramos acceder a estos recursos debemos de remplazar la línea `namespace :api do` dentro de `config/routes` que indica que todas las acciones dentro de este controlador van a responder al formato json

```ruby
namespace :api, defaults: {format: 'json'} do
  resources :users, only: [:index, :create, :destroy, :update, :show]
end
```

Al intentar acceder al recurso sin especificar el formato, el resultado debería de ser identico al anterior. http://localhost:3000/api/users (http://localhost:3000/api/users).

De igual forma al querer ver un 'user' http://localhost:3000/api/users/1 (http://localhost:3000/api/users/1).

Probar verbos POST, PUT y DELETE

Para probar las acciones update, create y destroy debemos de enviarle a nuestro recurso verbos diferentes a GET. Debemos de usar los verbos PUT, POST y DESTROY respectivamente. Para esto usaremos un programa de línea de comandos llamado curl. El cual podemos instalar con el siguiente comando:

```ruby
$ sudo apt-get install curl
```

Para probar el verbo CREATE usaremos:

```ruby
curl -X POST -d"user[username]=jesuslerma&user[email]=demo@desafio.com&user[password]=123" http
```

Para probar el verbo PUT

```ruby
curl -X PUT -d"user[username]=jesuslerma&user[email]=demo@desafio.com&user[password]=123" ht
```

Para probar el verbo DELETE

```ruby
curl -X DELETE http://localhost:3000/api/users/1
```

Así finalizamos con este tutorial. Si te gustó, no dudes en compartirlo, y dejarme tus comentarios por si tienes alguna duda.

Jesús Lerma mayo 24, 2016 4981 1

(http://blog.desafiolatam.com/category/uncategorized/)

(http://blog.desafiolatam.com/category/uncategorized/)APIs (http://blog.desafiolatam.com/category/tutoriales/rails/apis/), Rails (http://blog.desafiolatam.com/category/tutoriales/rails/), Tutoriales (http://blog.desafiolatam.com/category/tutoriales/)

También te puede interesar