

Guía 6: Patrón de diseño MVC usando scaffold

Introducción

Ruby on Rails sigue el patrón de arquitectura de software Modelo-Vista-Controlador, que separa los datos de la lógica de negocio, de una aplicación de su representación y el módulo encargado de gestionar los eventos y las comunicaciones.

En la siguiente guía se aprenderá a crear aplicaciones web, utilizando el framework Ruby on Rails mediante el generador de código **scaffold**, que permite tener las funcionalidades básicas de administración de un modelo, CRUD (Create, Read, Update, Delete) es común para cualquier sistema transaccional. La idea es también conocer lo que es el patrón de diseño MVC, ya que al hacer uso de scaffold este genera todo el código del modelo, la vista y el controlador de la aplicación.

Objetivos

- Comprender el uso de scaffold y de los métodos CRUD generados.
- Creación de una aplicación usando scaffold.
- Entender el patrón de diseño MVC.

Tiempo

- Una sesión de clase.

Requerimientos

Software	Hardware
Sistema Debian 9 virtualizado en Virtual box con: <ul style="list-style-type: none">• Ruby versión 2.4.1• Rails 5.4.1• Nodejs• Sublimetext	Computadora con características: <ul style="list-style-type: none">• Memoria ram mínimo 2GB• Procesador mínimo 2.1 GHz

Referencia

- Elicia. Scaffolding en RoR. <https://gist.github.com/Elicia/8530046>
- Ruby on Rails org, RailsGuide, The Rails command line. http://guides.rubyonrails.org/command_line.html
- Ruby on Rails org, RailsGuide, Active Record Migrations. http://guides.rubyonrails.org/active_record_migrations.html

Desarrollo

1. Crear un nuevo proyecto en Rails.

1.1 Crear un proyecto y ubicarse en su interior haciendo uso del terminal.

```
$ rails new App_Scaffold
```

1.2 Generar un scaffold, con una tabla llamada **Buy** con los campos:

```
$ rails generate scaffold Buy category:string description:text amount:decimal
```

Hay que mencionar algo importante para la creación de modelos, siempre se escriben con la primera letra en mayúscula y en singular; en cualquier otro caso el framework por defecto corrige la escritura, como el caso anterior el nombre del modelo será **Buy**.

Tipos de datos

Cuando se genera un scaffold se debe de pasar como parámetro los tipos que datos que contendrá la tabla de la base de datos donde se guardaran los datos de la aplicación, en la siguiente tabla se muestra los distintos tipos de datos y uso.

Tipos de datos en Rails

Nombre	Tipo de datos
:binary	Datos binarios
:boolean	Datos tipo booleano puede ser verdadero o falso
:date	Datos de tipo fecha
:datetime	Datos de tipo fecha y hora
:decimal	Datos de tipo decimal.
:float	Datos de tipo flotante
:integer	Datos de tipo entero
:string	Datos de tipo cadena de caracteres.
:text	Dato de tipo texto, parecido al tipo string pero este admite una mayor cantidad de caracteres.
:time	Dato de tipo tiempo.
:timestamp	Tipo de dato similar al datetime, se dice que son sinónimos pero con diferentes características

Cuando se utiliza scaffold para crear operaciones CRUD, este crea todos los archivos y métodos necesarios para el correcto funcionamiento del proyecto. Scaffold crea modelos, vistas, controladores, hojas de estilo, entre otros archivos y directorios como se muestra en la siguiente figura.

```
ismael@ismael:~/Escritorio/Proyecto_RoR$ rails new App_Scaffold
  create
  create  README.md
  create  Rakefile
  create  .ruby-version
  create  config.ru
  create  .gitignore
  create  .gitattributes
  create  Gemfile
  run     git init -b main from "."
Inicializado repositorio Git vacío en /home/ismael/Escritorio/Proyecto_RoR/App_Scaffold/.git/
  create  app
  create  app/assets/config/manifest.js
  create  app/assets/stylesheets/application.css
  create  app/channels/application_cable/channel.rb
  create  app/channels/application_cable/connection.rb
  create  app/controllers/application_controller.rb
  create  app/helpers/application_helper.rb
  create  app/jobs/application_job.rb
  create  app/mailers/application_mailer.rb
  create  app/models/application_record.rb
  create  app/views/layouts/application.html.erb
  create  app/views/layouts/mailer.html.erb
  create  app/views/layouts/mailer.text.erb
  create  app/views/pwa/manifest.json.erb
  create  app/views/pwa/service-worker.js
  create  app/assets/images
```

2. Migrar los datos.

Las migraciones son el modo más conveniente de cambiar el esquema de la base de datos a través del tiempo de una manera consistente y fácil, utilizan un lenguaje de definición de esquemas (DSL) en Ruby, por lo que no tiene que escribir SQL.

```
$ rake db:migrate
```

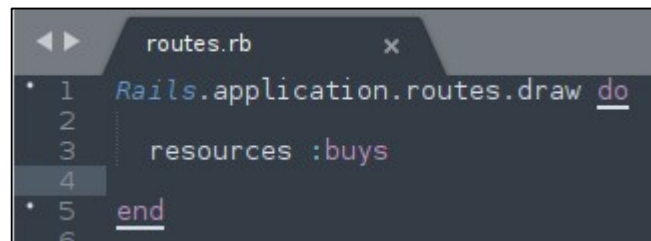
Al escribir el código anterior se obtiene la siguiente salida

```
ismael@ismael:~/Escritorio/Proyecto_RoR/App_Scaffold$ rake db:migrate
rbenv: version 'ruby-3.1.2' is not installed (set by /home/ismael/Escritorio/Proyecto_RoR/App_Scaffold/.ruby-version)
ismael@ismael:~/Escritorio/Proyecto_RoR/App_Scaffold$ sudo rake db:migrate
== 20240912035336 CreateBuys: migrating =====
-- create_table(:buys)
   -> 0.0035s
== 20240912035336 CreateBuys: migrated (0.0036s) =====
```

3. Rutas

Las rutas son una parte muy importante de la aplicación. Como se mencionó anteriormente scaffold genera las rutas, para que pueda existir una conexión entre las peticiones del usuario y la aplicación.

Una de las diferencias es que scaffold utiliza enrutamiento de recursos, esto permite declarar todas las rutas comunes para un controlador, en lugar de declarar rutas por separadas por cada una de las acciones que realiza el controlador como: index, show, new etc. Esta ruta se declara en una sola línea de código en el archivo **routes.rb**, como se observa en la figura 60. En este caso scaffold la genera automáticamente.

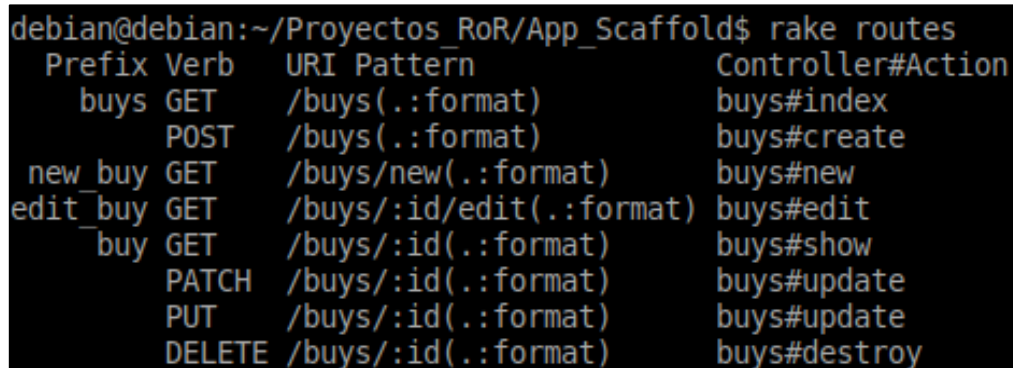


```
1 Rails.application.routes.draw do
2
3   resources :buys
4
5 end
```

3.1 Escribir el siguiente comando en el terminal y observar las rutas generadas.

```
$ rake routes
```

Observar que se muestran todas las rutas configuradas en el proyecto.



Prefix	Verb	URI Pattern	Controller#Action
buys	GET	/buys(:format)	buys#index
	POST	/buys(:format)	buys#create
new_buy	GET	/buys/new(:format)	buys#new
edit_buy	GET	/buys/:id/edit(:format)	buys#edit
buy	GET	/buys/:id(:format)	buys#show
	PATCH	/buys/:id(:format)	buys#update
	PUT	/buys/:id(:format)	buys#update
	DELETE	/buys/:id(:format)	buys#destroy

Cada ruta está compuesta por los siguientes elementos: verbos HTTP (GET, POST, PATCH, PUT, DELETE), path (camino), controlador y método (acción).

En Rails, una ruta proporciona una asignación entre los verbos HTTP y las URL a las acciones del controlador. Por defecto, cada acción también se asigna a operaciones CRUD particulares en la base de datos.

4. Verificar el controlador.

Observar que se han creado los distintos métodos del controlador en el archivo **buys_controller.rb** dentro del directorio (app/controller/).

```
app > controllers > buys_controller.rb
1 class BuysController < ApplicationController
2   before_action :set_buy, only: %i[ show edit update destroy ]
3
4   # GET /buys or /buys.json
5   def index
6     @buys = Buy.all
7   end
8
9   # GET /buys/1 or /buys/1.json
10  def show
11  end
12
13  # GET /buys/new
14  def new
15    @buy = Buy.new
16  end
17
18  # GET /buys/1/edit
19  def edit
20  end
21
22  # POST /buys or /buys.json
23  def create
24    @buy = Buy.new(buy_params)
25
26    respond_to do |format|
27      if @buy.save
28        format.html { redirect_to buy_url(@buy), notice: "Buy was successfully created." }
29        format.json { render :show, status: :created, location: @buy }
30      else
31        format.html { render :new, status: :unprocessable_entity }
32        format.json { render json: @buy.errors, status: :unprocessable_entity }
33      end
34    end
35  end
36
37  # PATCH/PUT /buys/1 or /buys/1.json
38  def update
39    respond_to do |format|
40      if @buy.update(buy_params)
41        format.html { redirect_to buy_url(@buy), notice: "Buy was successfully updated." }
42        format.json { render :show, status: :ok, location: @buy }
43      else
44        format.html { render :edit, status: :unprocessable_entity }
45        format.json { render json: @buy.errors, status: :unprocessable_entity }
46      end
47    end
48  end
49
```

5. Configurar el index generado por el scaffold como la página de inicio de la aplicación.

5.1 Escribir la siguiente línea de código en el archivo config/**routes.rb**.

```
root "buys#index"
```

```
$ rails server
```

5.2 Iniciar el servidor.

5.3 Abrir el navegador y acceder a la dirección.

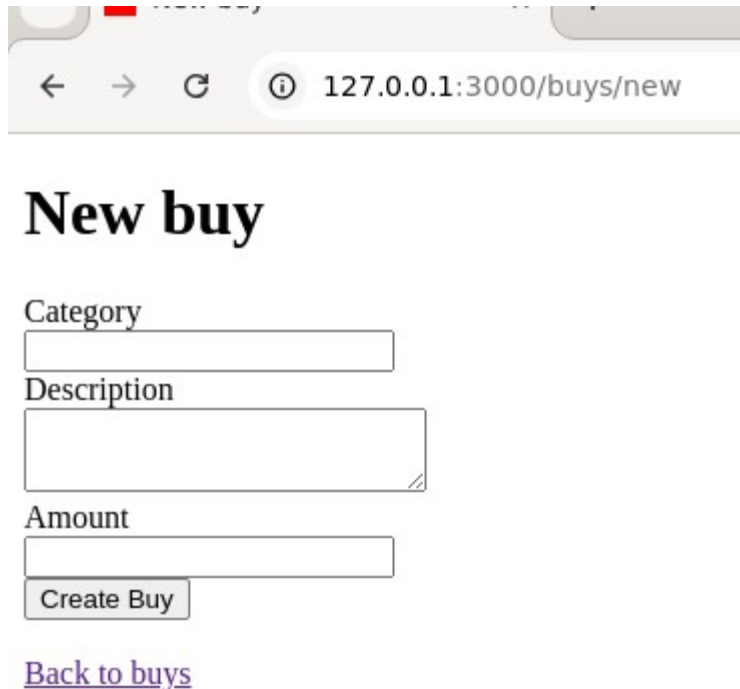
```
http://localhost:3000/
```



Se muestra la interfaz vacía, debido a que no se ha creado ningún dato y como se observa en la figura 63, este es el index generado por el scaffold, una vez configurado el archivo routes.rb se estableció como la página principal de la aplicación.

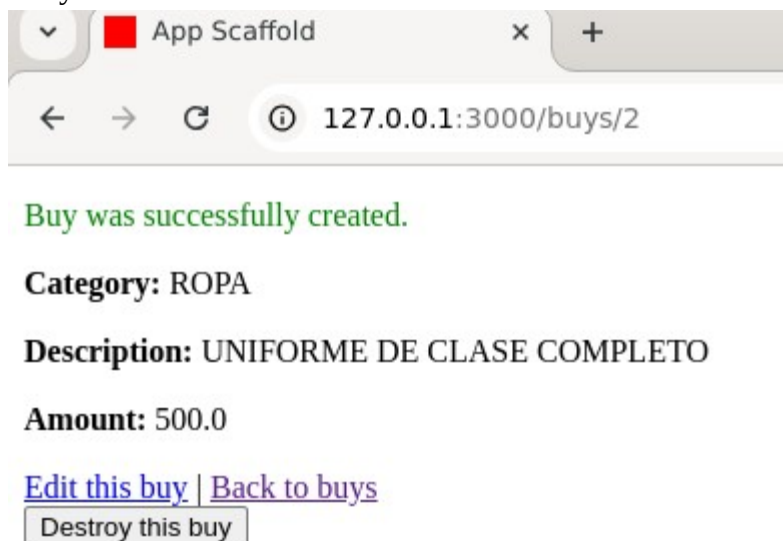
6. Verificar el funcionamiento de las vistas.

6.1 En la vista mostrada anteriormente, presiona new buy, se mostrará el formulario para poder agregar datos a la aplicación.



The screenshot shows a web browser window with the address bar displaying '127.0.0.1:3000/buys/new'. The page title is 'New buy'. Below the title, there is a form with three input fields: 'Category', 'Description', and 'Amount'. The 'Description' field is a larger text area. Below the 'Amount' field is a button labeled 'Create Buy'. At the bottom of the form, there is a link labeled 'Back to buys'.

6.2 Ingresar una nueva compra llenando el formulario, al presionar el botón **create buy**, el dato se ha creado y almacenado correctamente.



The screenshot shows a web browser window with the address bar displaying '127.0.0.1:3000/buys/2'. The page shows a confirmation message: 'Buy was successfully created.' Below this message, the details of the created buy are displayed: 'Category: ROPA', 'Description: UNIFORME DE CLASE COMPLETO', and 'Amount: 500.0'. At the bottom, there are two links: 'Edit this buy' and 'Back to buys', and a button labeled 'Destroy this buy'.

1.1 Click en **Edit**, podrá observar que permite editar el dato anteriormente almacenado.

Editing buy

1.2 Verificar que se

Category
ROPA

Description
UNIFORME DE CLASE
COMPLETO

Amount
500.0

[Show this buy](#) | [Back to buys](#)

muestran los datos almacenados dentro del index de la aplicación.

Category: ROPA

Description: UNIFORME DE CLASE COMPLETO

Amount: 500.0

[Edit this buy](#) | [Back to buys](#)

Puede interactuar con la aplicación agregando nuevos datos, de manera que pruebe las distintas funcionalidades creadas haciendo uso de scaffold.

2. Validaciones al modelo buy

Si el usuario agrega un nuevo dato en la aplicación, podrá agregar datos vacíos debido a que no se le ha agregado ningún tipo de validaciones al modelo. Una de las ventajas de Rails es permite validar los campos del formulario de una manera más sencilla, en este caso solo se validará que los campos del modelo no estén vacíos para poder agregar un nuevo dato a la aplicación.

2.1 Abrir el archivo `/model/buy.rb` y agregar el siguiente código.

```
validates :category, :description, :amount, presence: true
```

Si ahora intenta agregar un nuevo dato vacío le mostrará un conjunto de mensajes a como se muestra en la siguiente figura.

New buy

3 errors prohibited this buy from being saved:

- Category can't be blank
- Description can't be blank
- Amount can't be blank

Category

Description

Amount

Create Buy

[Back to buys](#)

Figura 1 Campos del buy validados

Ejercicios propuestos para ser entregados al docente

1. Realizar y analizar cada uno de los enunciados de la guía.
2. Crear un proyecto nuevo, utilice scaffold para generar el código y crear una tabla de nombre **Estudiante** con los campos nombres, apellidos, carrera, carnet, fecha de nacimiento, edad, celular; deberá configurar el archivo routes.rb para que la página principal de la aplicación sea el index generado por el scaffold, validar el campo celular para que solo admita número y que no permita campos vacíos. La aplicación deberá mostrar un formulario parecido al de la **figura 69**, se puede observar en la figura como el framework crea automáticamente las cajas de texto, dependiendo del tipo de dato que se le especifica al generar el scaffold, como el campo fecha de nacimiento donde crea un input tipo date_select en el formulario para seleccionar la fecha de nacimiento del estudiante.



New estudiante

← → ↻ ⓘ 127.0.0.1:3000/estudiantes/new

New estudiante

Nombres

Apellidos

Carrera

Carnet

Fecha de nacimiento
 

Edad

Celular

Create Estudiante

[Back to estudiantes](#)