

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 de 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.

```
invoke active_record
create db/migrate/20180216145848_create_buys.rb
create app/models/buy.rb
invoke test_unit
create test/models/buy_test.rb
create test/fixtures/buys.yml
invoke resource_route
route resources :buys
invoke scaffold_controller
create app/controllers/buys_controller.rb
invoke erb
create app/views/buys
create app/views/buys/index.html.erb
create app/views/buys/edit.html.erb
create app/views/buys/show.html.erb
create app/views/buys/new.html.erb
create app/views/buys/_form.html.erb
invoke test_unit
create test/controllers/buys_controller_test.rb
invoke helper
create app/helpers/buys_helper.rb
invoke test_unit
invoke jbuilder
create app/views/buys/index.json.jbuilder
create app/views/buys/show.json.jbuilder
create app/views/buys/_buy.json.jbuilder
invoke test_unit
create test/system/buys_test.rb
invoke assets
invoke coffee
create app/assets/javascripts/buys.coffee
invoke scss
create app/assets/stylesheets/buys.scss
invoke scss
create app/assets/stylesheets/scaffolds.scss
```

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
```

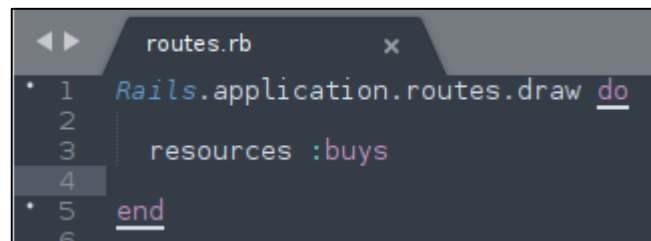
Al escribir el código anterior se obtiene la siguiente salida

```
debian@debian:~/Proyectos_RoR/App_Scaffold$ rake db:migrate
== 20180216145848 CreateBuys: migrating =====
-- create table(:buys)
-> 0.0023s
== 20180216145848 CreateBuys: migrated (0.0025s) =====
```

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.

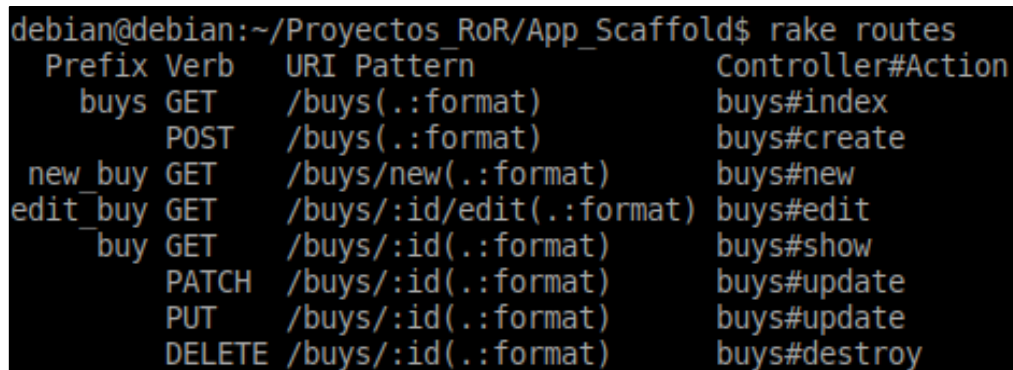


```
1 Rails.application.routes.draw do
2
3   resources :buys
4
5 end
```

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.



```
debian@debian:~/Proyectos_RoR/App_Scaffold$ rake routes
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/).

```
buys_controller.rb x
1 class BuysController < ApplicationController
2   before_action :set_buy, only: [:show, :edit, :update, :destroy]
3
4   # GET /buys
5   # GET /buys.json
6   def index
7     @buys = Buy.all
8   end
9
10  # GET /buys/1
11  # GET /buys/1.json
12  def show
13  end
14
15  # GET /buys/new
16  def new
17    @buy = Buy.new
18  end
19
20  # GET /buys/1/edit
21  def edit
22  end
23
24  # POST /buys
25  # POST /buys.json
26  def create
27    @buy = Buy.new(buy_params)
28
29    respond_to do |format|
30      if @buy.save
31        format.html { redirect_to @buy, notice: 'Buy was successfully created.' }
32        format.json { render :show, status: :created, location: @buy }
33      else
34        format.html { render :new }
35        format.json { render json: @buy.errors, status: :unprocessable_entity }
36      end
37    end
38  end
39
40  # PATCH/PUT /buys/1
41  # PATCH/PUT /buys/1.json
42  def update
43    respond_to do |format|
44      if @buy.update(buy_params)
45        format.html { redirect_to @buy, notice: 'Buy was successfully updated.' }
46        format.json { render :show, status: :ok, location: @buy }
47      else
48        format.html { render :edit }
49        format.json { render json: @buy.errors, status: :unprocessable_entity }
50      end
51    end
52  end
53
54  def destroy
55    @buy.destroy
56    respond_to do |format|
57      format.html { redirect_to buys_url, notice: 'Buy was successfully destroyed.' }
58      format.json { head :no_content }
59    end
60  end
61
62  private
63
64  # Use callbacks to share common setup or constraints between actions.
65  def set_buy
66    @buy = Buy.find(params[:id])
67  end
68
69  # Never trust parameters from the scary internet, only allow the white list through.
70  def buy_params
71    params.require(:buy).permit(:category, :description, :amount)
72  end
73
74 end
```

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**.

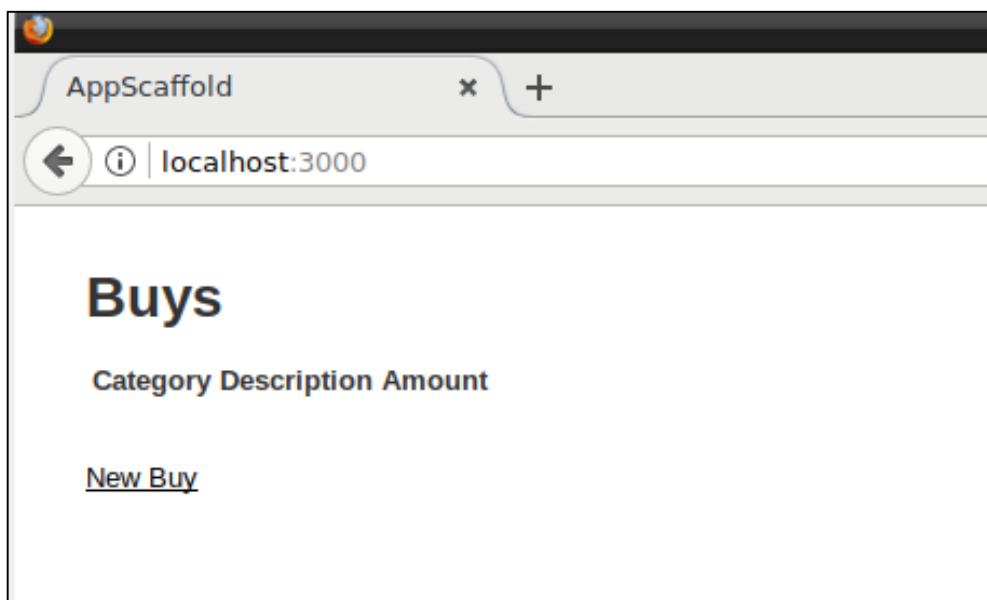
```
root "buys#index"
```

```
$ rails server
```

5.2 Iniciar el servidor.

5.3 Abrir el navegador y acceder a la dirección.

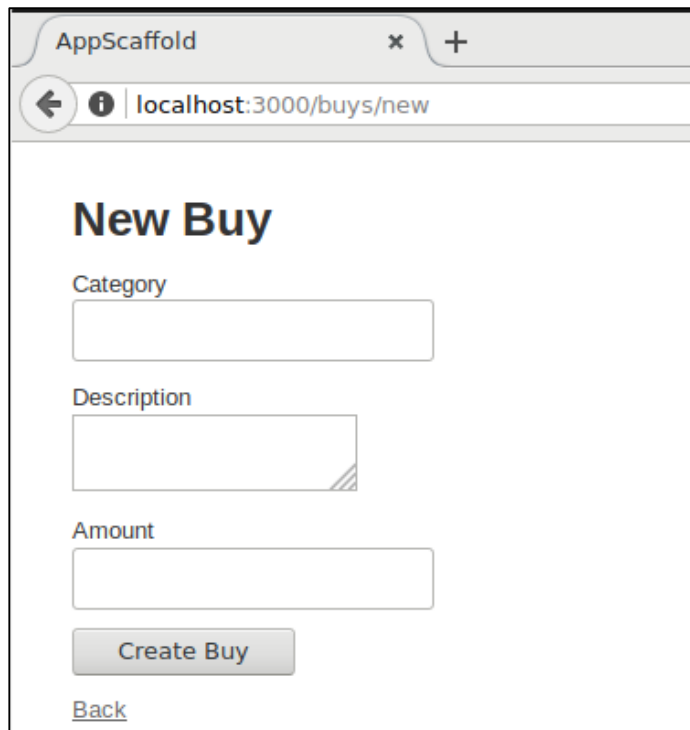
```
http://localhost:3000/
```



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.

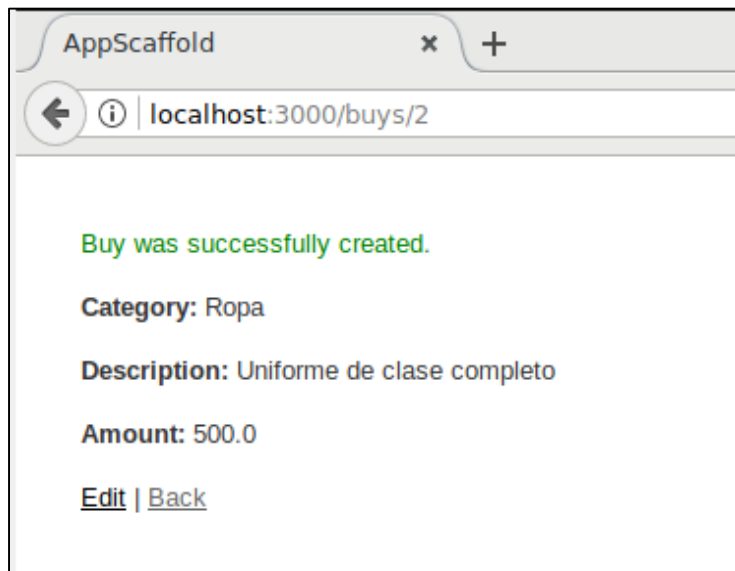
6. Verificar el funcionamiento de las vistas.

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



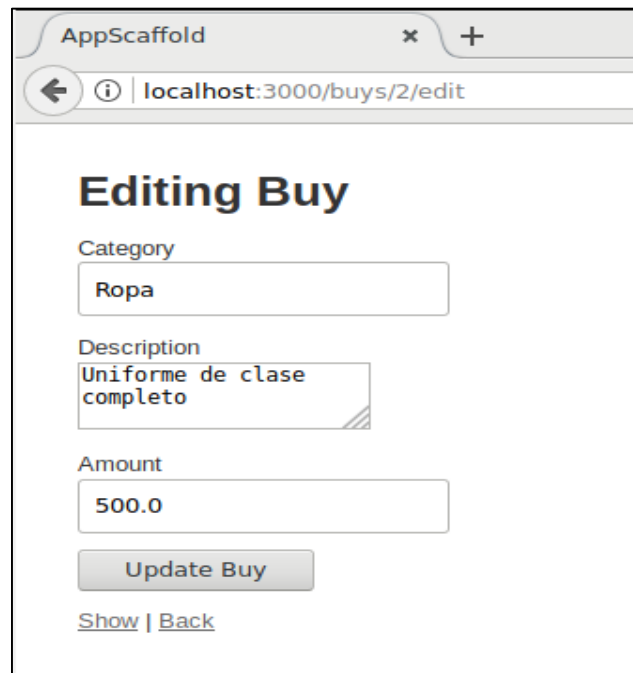
The screenshot shows a web browser window with the title 'AppScaffold'. The address bar displays 'localhost:3000/buys/new'. The main content area is titled 'New Buy' and contains three input fields: 'Category', 'Description', and 'Amount'. Below these fields is a 'Create Buy' button and a 'Back' link.

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



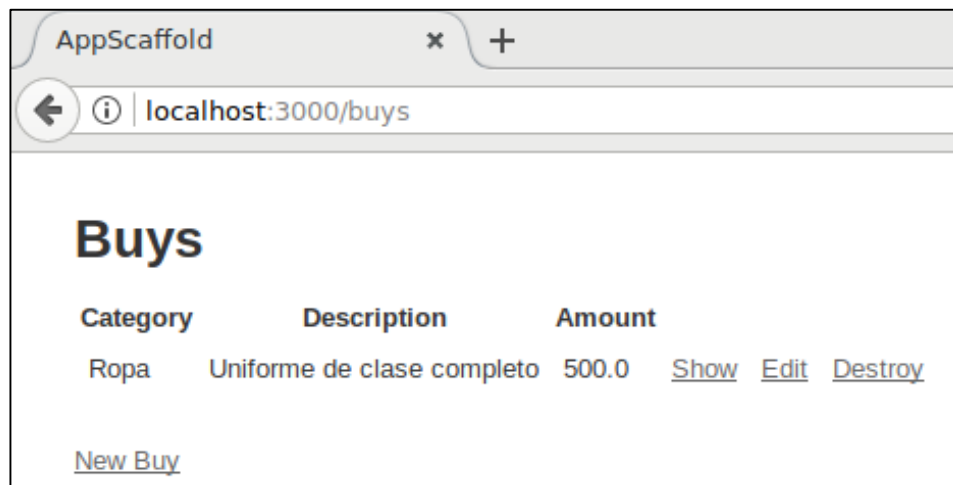
The screenshot shows a web browser window with the title 'AppScaffold'. The address bar displays 'localhost:3000/buys/2'. The main content area displays the following information: 'Buy was successfully created.', 'Category: Ropa', 'Description: Uniforme de clase completo', and 'Amount: 500.0'. At the bottom, there are links for 'Edit' and 'Back'.

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



The screenshot shows a web browser window with the title 'AppScaffold'. The address bar displays 'localhost:3000/buys/2/edit'. The main content area is titled 'Editing Buy' and contains three input fields: 'Category' with the value 'Ropa', 'Description' with the value 'Uniforme de clase completo', and 'Amount' with the value '500.0'. Below these fields is a button labeled 'Update Buy'. At the bottom, there are two links: 'Show' and 'Back'.

1.2 Verificar que se muestran los datos almacenados dentro del index de la aplicación.



The screenshot shows a web browser window with the title 'AppScaffold'. The address bar displays 'localhost:3000/buys'. The main content area is titled 'Buys' and contains a table with the following data:

Category	Description	Amount	
Ropa	Uniforme de clase completo	500.0	Show Edit Destroy

Below the table, there is a link labeled 'New Buy'.

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

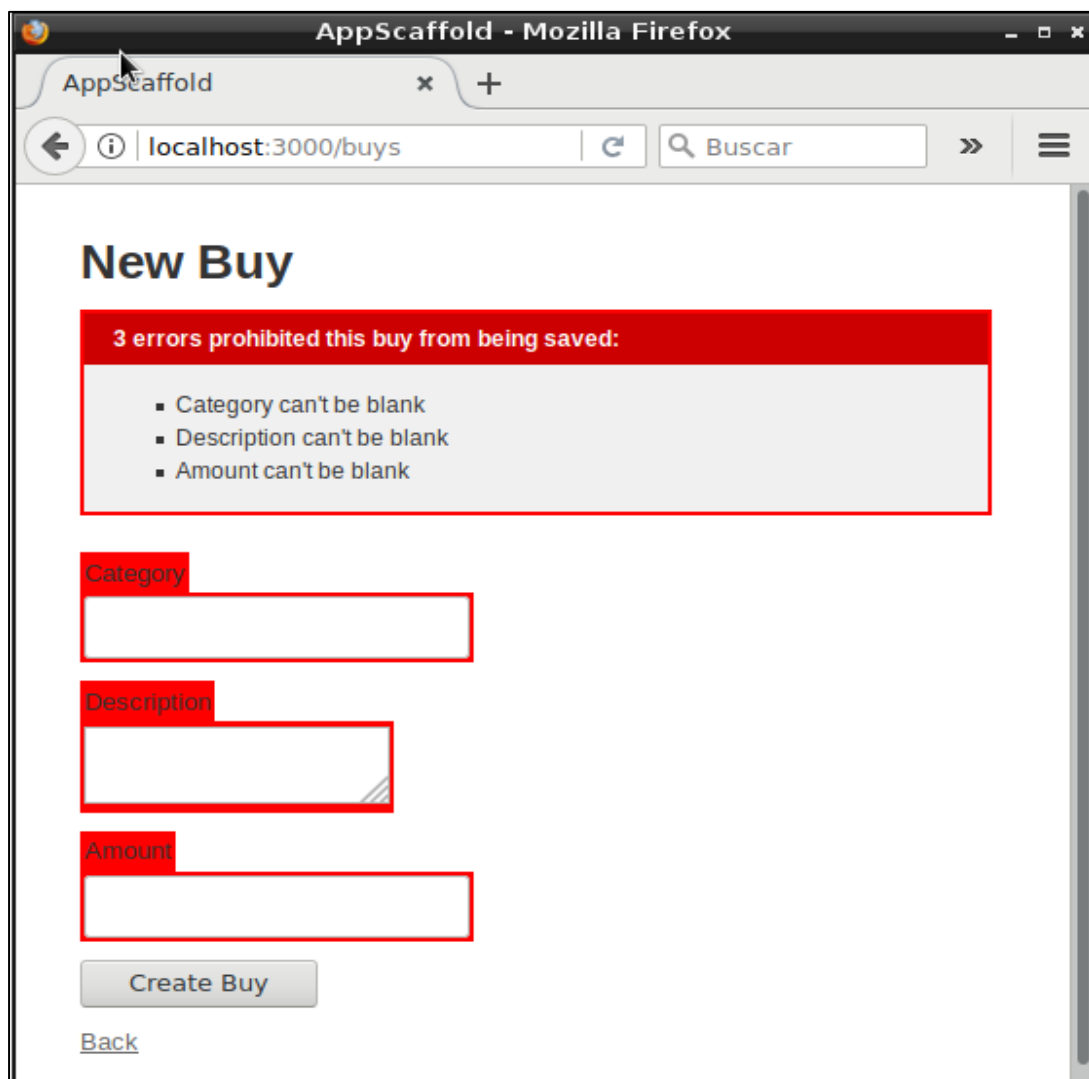
2. Validaciones al modelo buy

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.

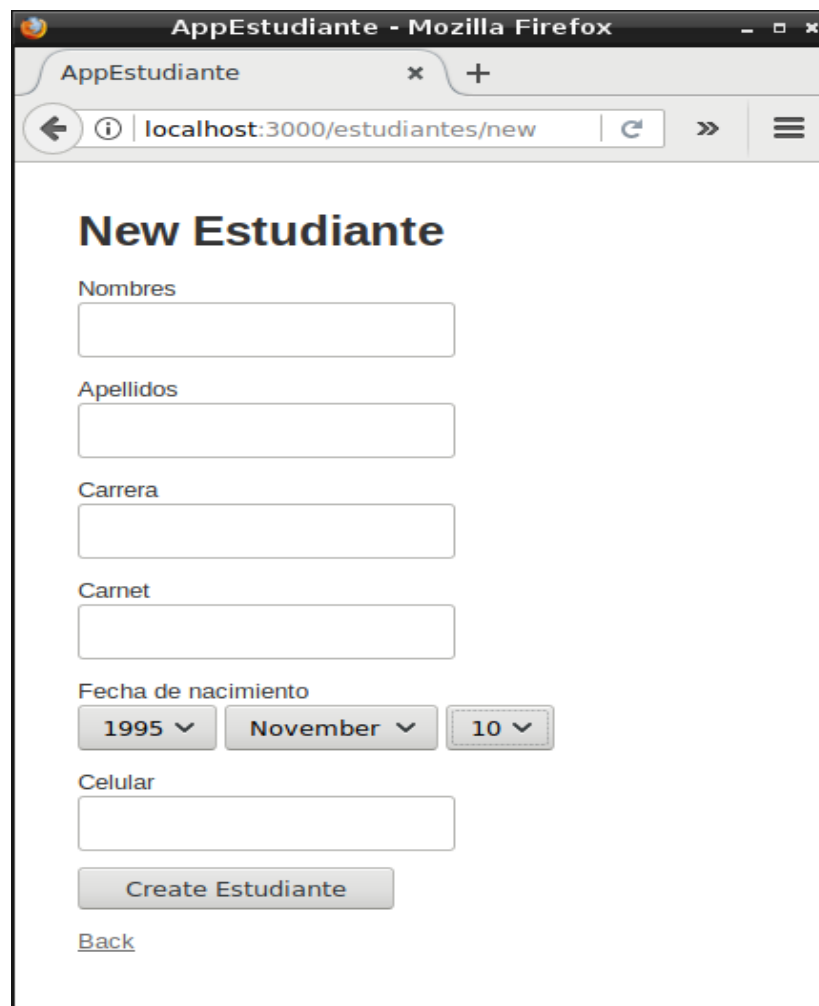


The screenshot shows a web browser window titled 'AppScaffold - Mozilla Firefox' with the address bar at 'localhost:3000/buys'. The page displays a 'New Buy' form. At the top of the form, a red banner states '3 errors prohibited this buy from being saved:'. Below this, a list of errors is shown: 'Category can't be blank', 'Description can't be blank', and 'Amount can't be blank'. The form contains three input fields: 'Category', 'Description', and 'Amount', all of which are empty. Below the input fields is a 'Create Buy' button and a 'Back' link.

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.



The screenshot shows a web browser window titled 'AppEstudiante - Mozilla Firefox'. The address bar displays 'localhost:3000/estudiantes/new'. The page content is a form titled 'New Estudiante'. The form includes the following fields and controls:

- Nombres**: A text input field.
- Apellidos**: A text input field.
- Carrera**: A text input field.
- Carnet**: A text input field.
- Fecha de nacimiento**: A date select control with three dropdown menus showing '1995', 'November', and '10'.
- Celular**: A text input field.
- Create Estudiante**: A button to submit the form.
- Back**: A link to navigate back.