Rails: revisión de lo esencial

Crear una nueva aplicación llamada simple

% rails generate simple

- Genera toda la estructura de directorios y archivos necesaria

% bundle install y/o bundle update

- El primero maneja cambios en archivo de gemas
- El segundo actualiza gemas que ya están siendo manejadas

Creación de un controlador

% rails generate controller hello

- Genera un archivo hello_controller.rb en la carpeta donde viven los controllers
- El archivo contiene una clase: class HelloController < Application Controller
- No tiene métodos aún

Agregamos un método al controlador que llamaremos "in"

- Este método lo asociaremos a la acción in que manejará el controller
- Acción in quedará asociada auna url mediante archivo de rutas
- Lo unico que hará el controlador será cargar la variable @nombre con el string 'Jaime'

```
class HelloController < ApplicationController
  def in
    @nombre = 'Jaime'
  end
end
```

Agregamos un segundo método

En general un controlador maneja muchas acciones

- escribiremos un método out

```
class HelloController < ApplicationController
  def in
    @nombre = 'Jaime'
  end
  def out
    @nombre = 'Jaime'
  end
end
```

Necesitamos agregar una vista para cada una de esas acciones

- El controlador pasa la acción a la vista una vez que completa su trabajo

Las Vistas

De acuerdo a la convención deben llevar nombres in y out como las acciones del controlador respectivo

views/hello/in.html.erb

views/hello/out.html.erb

```
<h1>Hello
  <%= @nombre %>
</h1>
```

```
<h1>Bye
  <%= @nombre %>
</h1>
<h2> See you later </h2>
```

Asociación de la ruta

Archivo config/routes.rb

- agregamos una línea indicando que un GET /hello/in debe activar el método in del controlador hello
- agregamos una línea indicando que un GET /hello/out debe activar el método in del controlador hello
- agregamos además que la página raiz GET / también debe llevar a lo mismo

```
Rails.application.routes.draw do
  root "hello#in"
  get '/hello/in', to: 'hello#in'
  get '/hello/out', to: 'hello#out'
end
```

Levantar el servidor local y probar

% rails server

- luego de unos segundos el server indicará que está escuchando solicitudes http en el puerto 3000
- pueden observar aquí mismo lo que va sucediendo con las solicitudes hechas desde el browser

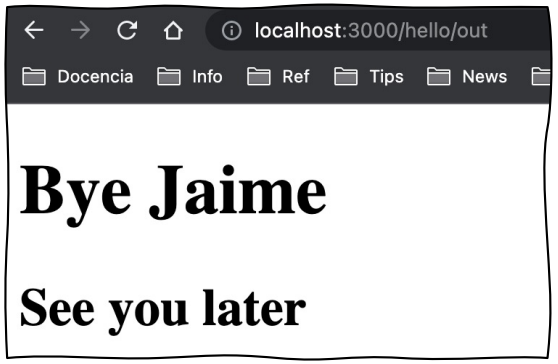
Con nuestro navegador favorito buscamos la url

- localhost:3000/
- localhost:3000/hello/in
- localhost:3000/hello/out

localhost:3000/

localhost:3000/hello/in

localhost:3000/hello/out



Agregamos un CRUD

Un CRUD son operaciones de creación, lectura, actualización y eliminación de un objeto dado

La manera más sencilla es usar la creación automática de scaffoldings de rails

- crea el controlador y sus acciones
- crea las vistas asociadas
- crea el modelo de ese objeto
- crea las rutas

Student

Utilizando la facilidad de scaffolding

% rails generate scaffolding Student rut:string name:string

- un objeto de la clase Student tiene solo dos atributos: rut y name
- rails creará una tabla en la base de datos llamada students
- cada objeto de esa clase corresponderá a una fila de la tabla

Para crear la base de datos debemos correr

% rails db:migrate

- en realidad el generate anterior crea la migración pero no la BD

El controlador creado: students_controller.rb

```
class StudentsController < ApplicationController

# GET /students or /students.json
def index
@students = Student.all
end

# GET /students/1 or /students/1.json
def show
end

# GET /students/new
def new
@student = Student.new
end

# GET /students/1/edit
def edit
end

# POST /students or /students.json
def create
@student = Student.new(student_params)
…
end

# PATCH/PUT /students/1 or /students/1.json
def update
…
end

# DELETE /students/1 or /students/1.json
def destroy
@student.destroy
…
end

end
```

Las vistas creadas

index.html.erb
- despliega todos los estudiantes

edit.html.erb
- para editar un estudiante

new.html.erb
- para agregar un estudiante

show.html.erb
- para mostrar un estudiante

Se crean trozos reusables (partials)

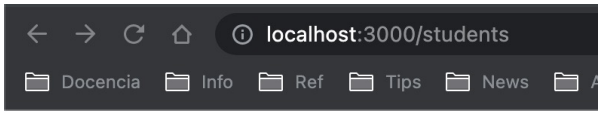
_student.html.erb
_form_html.erb

El archivo de rutas solo agrega una línea !

```
Rails.application.routes.draw do
resources :students
root "hello#in"
get '/hello/in', to: 'hello#in'
get '/hello/out', to: 'hello#out'
end
```

resource permite asociar todas las acciones con las vistas y los caminos

Probando ...



Students

Rut: 123-4

Name: Juan Perez

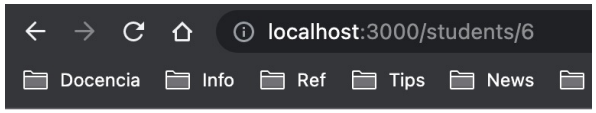
Show this student

Rut: 324-6

Name: Jaime Navon

Show this student

New student

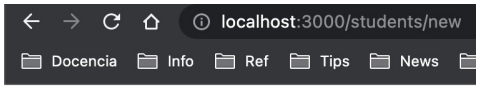


Rut: 324-6

Name: Jaime Navon

Edit this student | Back to students

Destroy this student



New student

Rut

Name

Create Student

Back to students

El Modelo ?

Sorprendentemente lo único que encontramos en la carpeta de modelos es un archivo student.rb con una clase que pareciera no tener nada

```
class Student < ApplicationRecord
end
```

Recordemos que Student es una subclase de ApplicationRecord por lo que hereda todo lo necesario

Hay dos cosas que suelen agregarse

- reglas de validación
- asociaciones con otras entidades

Reglas de validación

Por ejemplo supongamos que no aceptamos que nos ingresen un estudiante sin especificar el rut

- basta agregar "validates: rut, presence: true" a la clase Student

```
class Student < ApplicationRecord
  validates :rut, presence: true
end
```

- si queremos especificar un largo máximo de 30 para el nombre

```
class Student < ApplicationRecord
  validates :rut, presence: true
  validates :name, length: {maximum: 30}
end
```

Asociando con otras entidades

Supongamos que los alumnos tienen publicaciones en un blog

- un alumno puede tener 0, 1 o más publicaciones
- una publicación pertenece a solo un alumno
- cada publicación tiene solo dos atributos: el contenido y el alumno que lo publicó

Las tablas asociadas a las entidades (Clases de ActiveRecord) siempre tienen una columna id que es la clave (es un tipo de dato de tipo autoincrementado)

- el primer alumno quedará con id 1, el segundo con 2 y así
- las publicaciones hacen referencia a este número

Creación de las publicaciones

% rails generate scaffolding Pub content:text student_id:integer

% rails db:migrate

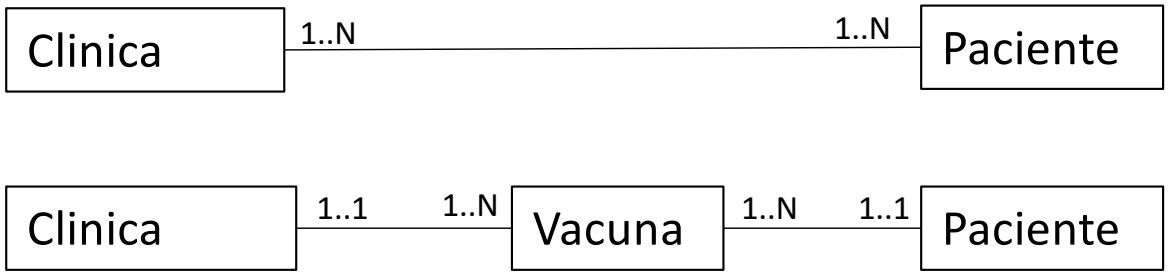
Ahora es necesario enlazar las entidades Student y Pub

```
class Student < ApplicationRecord
validates :rut, presence: true
has_many :pubs
end
```

```
class Pub < ApplicationRecord
belongs_to :student
end
```

Otras asociaciones

- belongs_to
- has_one
- has_many
- has_and_belongs_to_many
- has_many :through
- has_one :through



Las dos últimas permiten generar asociaciones a través de una entidad intermedia

- una clínica vacuna a muchos pacientes, un paciente puede usar varias clínicas
- la vacuna misma puede asociar a una clínica con un paciente

Métodos interesantes de ActiveRecord

Gracias a eso podemos acceder fácilmente a la información de las tablas (que corresponden a instancias de Student y Pub)

- Student.all - todos los objetos de la clase Student
- Student.first – el primer objeto
- Student.find(3) – el objeto con id = 3
- Student.find_by(name: 'Jaime Navon') – el objeto con ese nombre
- x = Student.find(1); x.pubs – todas las publicaciones del estudiante 1
- y = Pubs.find(1); y.student – el autor de la publicacion 1