TALLER DE SISTEMAS DE COMPUTACIÓN
2016-2

RAILS + API REST

INTEGRANTES:
ROBERTO FUENTES
CARLOS JAUREGUI
PATRICIO SARD

PROFESOR:
EUGENIO CANALES

CREACIÓN DE PROYECTO

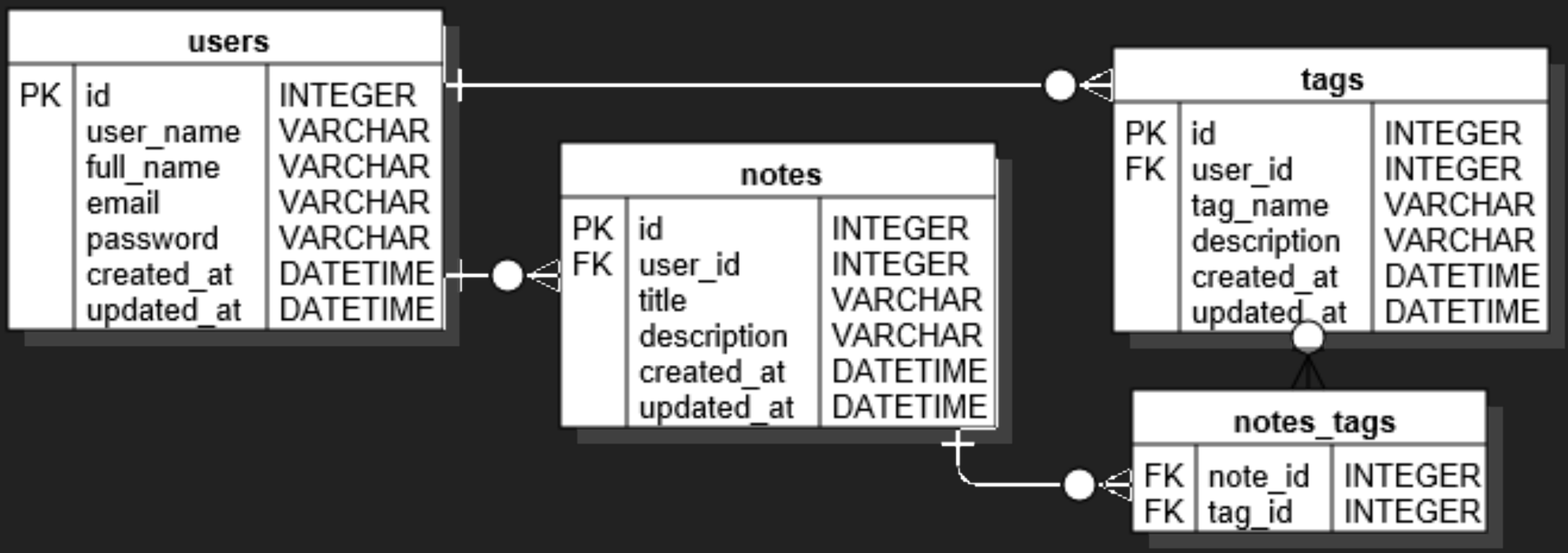
CREACIÓN DEL PROYECTO

- Primero creamos el proyecto con el comando `rails new tsc-grupo2-api --api`. Este ultimo es una opción de rails 5 en adelante, para crear una aplicación con todas las características de una api.

```
[root@ip211 home]# rails new tsc-grupo2-api --api
```

CREACIÓN DE MODELO, CONTROLADORES Y TABLAS

- Para la creación de la tabla con los modelos y controladores asociados para un recurso usamos el comando `scaffold`. Replicaremos entonces el modelo de datos subido al principio del curso.



CREACIÓN DE MODELO, CONTROLADORES Y TABLAS

- El comando `rails g` nos permite generar plantillas con varios elementos por defecto, tales como modelos, controladores, etc, automatizando la creación de funcionalidades y/o recursos.

- Para la creación de usuario:

```
[root@ip211 tsc-grupo2-api]# rails g scaffold User user_name:string full_name:string email:string password:string
```

- Para la creación de etiquetas:

```
[root@ip211 tsc-grupo2-api]# rails g scaffold Tag user:belongs_to tag_name:string description:string
```

- Para la creación de notas:

```
[root@ip211 tsc-grupo2-api]# rails g scaffold Note user:belongs_to title:string description:string
```

- De esta forma, crearemos todo el CRUD para los recursos de etiquetas, notas y usuarios. De forma automática, se crearan las rutas, controladores y tablas en la base de datos necesarias para la creación de nuestros registros.

USO DE BELONGS_TO

- Es importante explicar la opción `user:belongs_to` en los pantallazos mostrados anteriormente:

```
[root@ip211 tsc-grupo2-api]# rails g scaffold Tag user:belongs_to tag_name:string description:string
```

- Con esta opción la tabla tags pondremos atributo de referencia a user llamada `user_id` y en el modelo Tag se pondrá de forma automática el comando `belongs_to :user`, con el cual una instancia del modelo Tag podrá usar métodos como .user para que retorne el usuario del cual pertenece la etiqueta.

- Ahora falta especificar que un usuario puede tener varios etiquetas y notas asociadas a el. Esto lo haremos agregando los siguientes comandos en el archivo `user.rb`, que se encuentra en la ruta `/app/models/`.

```
class User < ApplicationRecord
  has_many :tags
  has_many :notes
```

USO DE BELONGS_TO

- Para crear la relación muchos-muchos, se agrega las lineas `has_and_belongs_to_many :notes` en el archivo `tag.rb`, y `has_and_belongs_to_many :tags` en el archivo `note.rb`, ambos en la ruta `/app/models/`:

```ruby
class Tag < ApplicationRecord
  belongs_to :user
  has_and_belongs_to_many :notes

  validates :tag_name, presence: true, o
  validates :description, presence: true
```

```ruby
class Note < ApplicationRecord
  belongs_to :user
  has_and_belongs_to_many :tags

  validates :title, presence: true, on: :create
  validates :description, presence: true, on: :create
end
```

- Ademas, hay que crear una tabla, la cual contendrá por cada registro 2 claves id foráneas que asociaran una nota y una etiqueta. Esto lo haremos creando una tabla con el comando `rails g migration CreateNotesTags`, y dentro de este archivo poniendo lo siguiente:

```ruby
class CreateNotesTags < ActiveRecord::Migration[5.0]
  def change
    create_table :notes_tags, id: false do |t|
      t.belongs_to :note, index: true
      t.belongs_to :tag, index: true
    end
  end
end
```

REALIZAR CAMBIOS EN LA BD

- Una vez hecho todo esto, procedemos a realizar el comando `rake db:migrate` para que se produzcan todos los cambios respectivos en la base de datos.

```
[root@ip211 tsc-grupo2-api]# rake db:migrate
```

ASIGNAR ETIQUETA A UNA NOTA

- Luego, lo que debemos crear es la funcionalidad de asignar una etiqueta a una nota. Para hacer esto, iremos al controlador `notes_controller.rb` la cual se encuentra en la ruta `/app/controllers/`, y agregaremos la siguiente acción:

```ruby
# POST /notes/1/assign_tag/
def assign_tag
  unless @note.tags.exists?(params[:tag_id])
    @note.tags << Tag.find(params[:tag_id])

    render json: {status: :ok}
  else
    render json: {status: :not_modified}
  end
end
```

- Ademas, en `before_acction :set_note` agregaremos `:assign_tag`.

```ruby
class NotesController < ApplicationController
  before_action :set_note, only: [:show, :update, :destroy, :assign_tag]
```

- Finalmente, agregamos la ruta de esta acción:

```ruby
post 'notes/:id/assign_tag/' => 'notes#assign_tag'
```

MOSTRAR TODAS LAS ETIQUETAS Y NOTAS DE UN USUARIO

- Para esto, agregaremos la siguiente acción en el archivo `users_controller.rb`, el cual se encuentra en `/app/controllers`:

```ruby
# GET /users/1/notes_tags
def notes_tags
  @notes = @user.notes
  @tags = @user.tags

  render json: {notes: @notes, tags: @tags}
end
```

- Ademas, en `before_acction :set_note` agregaremos `:notes_tags`.

```ruby
class UsersController < ApplicationController
  skip_before_action :basic_authentication, only: :create
  before_action :set_user, only: [:show, :update, :destroy, :notes_tags]
```

- Finalmente, agregamos la ruta de esta acción:

```ruby
get 'users/:id/notes_tags/' => 'users#notes_tags'
```

VALIDACIONES

- Es importante destacar que para que en las consultas se evite el hecho de mandar parámetros nulos, o no enviar parámetros, agregaremos validaciones para asegurarnos que todos los registros tengan algún valor en todos sus atributos, y ademas para que los mails sean únicos. Esto se agrega en el archivo `user.rb` que se encuentra en `/app/models/`.

```
validates :user_name, presence: true, on: :create
validates :full_name, presence: true, on: :create
validates :email, presence: true, on: :create
validates :password, presence: true, on: :create

validates :email, uniqueness: true
```

AUTENTICACIÓN BASICA

AUTENTICACIÓN BÁSICA

- Para agregar una autenticación básica, agregaremos el modulo Base64, la cual provee funcionalidades para codificar y decodificar un `string`, y así decodificar las credenciales una vez recibidas. Esto se hará en el archivo `application.rb` en la carpeta `/config/`

```
require_relative 'boot'

require "rails"
# Pick the frameworks you want:
require "base64"
require "active_model/railtie"
require "active_job/railtie"
require "active_record/railtie"
```

MÉTODO DE AUTENTICACIÓN BÁSICA

- Ahora crearemos una acción que se ejecutara antes de cualquier otra, y que nos servirá para verificar que las credenciales que se envían codificadas en el header de la petición corresponden a los de un usuario en la base de datos. De no ocurrir esto, no se le permitirá hacer nada. Esto se hará en el archivo `application_controller.rb` que se encuentra en `/app/controller/`.

```ruby
class ApplicationController < ActionController::API
  before_action :basic_authentication

  def basic_authentication
    access = false


    if request.headers['Authorization']
      begin
        auth = Base64.strict_decode64(request.headers['Authorization'])
        auth = auth.split(':')
        user = User.find_by(email: auth[0])

        if user && user.password == auth[1]
          access = true
        end
      rescue ArgumentError => e
        render json: {status: :internal_server_error}

        return
      end
    end

    unless access
      render json: {status: :unauthorized}
    end
  end
end
```

AUTENTICACIÓN BÁSICA

- Para evitar que se necesite una autenticación al momento de registrarse, pondremos la siguiente linea de código en el archivo `users_controller.rb`, el cual se encuentra en la ruta `/app/controller/`:

```
skip_before_action :basic_authentication, only: :create
```

FIN