Práctica 4: Ruby on Rails

Taller de Tecnologías de Producción de Software - Ruby

Cursada 2023

En esta práctica veremos con cierto nivel de profundidad el *framework* Ruby on Rails, una excelente herramienta para el desarrollo ágil de aplicaciones y servicios web.

Ruby on Rails

1. El framework está compuesto de diferentes librerías:

   - ActionMailer
   - ActionPack
   - ActionView
   - ActiveJob
   - ActiveModel
   - ActiveRecord
   - ActiveSupport
   - ActionCable
   - ActiveStorage
   - ActionText
   - ActionMailbox

   Para cada una de estas librerías, analizá y respondé las siguientes preguntas:

   1. ¿Qué funcion principal cumple?
   2. Citá algún ejemplo donde se te ocurra que podrías utilizarla.

2. Investigá cómo se crea una aplicación Rails nueva y enumerá los pasos básicos para tener la aplicación funcionando con una base de datos SQLite.

   > Tip: investigá las opciones disponibles del comando `rails new`.

3. Siguiendo los pasos que enumeraste en el punto anterior, creá una nueva aplicación Rails llamada `practica_rails` en la cual vas a realizar las pruebas para los ejercicios de esta práctica.

4. ¿Qué es un ambiente (`environment`) en una aplicación Rails? ¿Qué sentido considerás que tiene usar diferentes ambientes en tu aplicación? ¿Cuál es el ambiente por defecto?

5. Sobre la configuración de Rails:

   1. ¿De qué forma podés especificar parámetros de configuración del framework en una app Rails?
   2. ¿Cuáles son los archivos de configuración principales? Intentá relacionar este concepto con los ambientes que antes viste.

3. Modificá el `locale` por defecto de tu aplicación para que sea español.
4. Modificá la zona horaria de tu aplicación para que sea la de la Argentina (GMT-3).

6. Sobre los *initializers*:

    1. ¿Qué son y para qué se utilizan?
    2. ¿En qué momento de la vida de la aplicación se ejecutan?
    3. Si tu app está corriendo y modificás algún initializer, ¿los cambios se reflejan automáticamente? ¿Por qué?
    4. Creá un initializer en tu aplicación que imprima en pantalla el string `"Booting practica_rails :)"`. ¿En qué momento se visualiza este mensaje?

7. Sobre los *generators*:

    1. ¿Qué son? ¿Qué beneficios imaginás que trae su uso?
    2. ¿Con qué comando podés consultar todos los generators disponibles en tu app Rails?
    3. Utilizando el generator adecuado, creá un controlador llamado `PoliteController` que tenga una acción `salute` que responda con un saludo aleatorio de entre un arreglo de 5 diferentes, como por ejemplo *"Good day sir/ma'am."*.

8. Sobre *routing*:

    1. ¿Dónde se definen las rutas de la app Rails?
    2. ¿De qué formas se pueden definir las rutas? Investigá la DSL para definición de rutas que Ruby on Rails provee.
    3. ¿Qué ruta(s) agregó el generator que usaste en el último inciso del punto anterior?
    4. ¿Con qué comando podés consultar todas las rutas definidas en tu aplicación Rails?
    5. Modificá el mapeo de rutas de tu aplicación para que al acceder a `/` (*root*) se acceda al controlador definido antes (`polite#salute`).

Componentes principales del framework

ActiveSupport (AS)

9. ¿De qué forma extiende AS las clases básicas de Ruby para incorporar nuevos métodos?
10. Investigá qué métodos agrega AS a las siguientes clases:

    1. `String`
    2. `Array`
    3. `Hash`

    4. `Date`
    5. `Numeric`

11. ¿Qué hacen y en qué clase define AS los siguientes métodos?

    1. `blank?`
    2. `present?`
    3. `presence`
    4. **`try`**
    5. `in?`
    6. `alias_method_chain`
    7. `delegate`
    8. `pluralize`
    9. `singularize`
    10. `camelize`
    11. `underscore`
    12. `classify`
    13. `constantize`
    14. `safe_constantize`
    15. `humanize`
    16. `sum`
    17. `with_indifferent_access`

12. ¿De qué manera se le puede *enseñar* a AS cómo pasar de singular a plural (o viceversa) los sustantivos que usamos en nuestro código?

    > Tip: Mirá el archivo `config/initializers/inflections.rb`.

13. Modificá la configuración de la aplicación Rails para que aprenda a pluralizar correctamente en español todas las palabras que terminen en `l`, `n` y `r`.

    > Tip: el uso de expresiones regulares simples ayuda.

ActiveRecord (AR)

14. ¿Cómo se define un modelo con ActiveRecord? ¿Qué requisitos tienen que cumplir las clases para utilizar la lógica de abstracción del acceso a la base de datos que esta librería ofrece?
15. ¿De qué forma *sabe* ActiveRecord qué campos tiene un modelo?
16. ¿Qué metodos (*getters* y *setters*) genera AR para los campos escalares básicos (`integer`, `string`, `text`, **`boolean`**, **`float`**)?

17. ¿Qué convención define AR para inferir los nombres de las tablas a partir de las clases Ruby? Citá ejemplos.
18. Sobre las migraciones de AR:
    1. ¿Qué son y para qué sirven?
    2. ¿Cómo se generan?
    3. ¿Dónde se guardan en el proyecto?
    4. ¿Qué formato/organización tienen sus nombres de archivo? ¿Por qué considerás que es necesario respetar ese formato?
    5. Generá una migración que cree la tabla `offices` con los siguientes atributos:
        - `name`: `string` de `255` caracteres, no puede ser nulo.
        - `phone_number`: `string` de `30` caracteres, no puede ser nulo y debe ser único.
        - `address`: `text`.
        - `available`: **`boolean`**, por defecto **`true`**, no puede ser nulo.
19. Creá el modelo `Office` para la tabla `offices` que creaste antes, e implementá en éste el método de instancia `to_s`.

    > Tip: también podés usar un generator para esto.

20. Utilizando migraciones, creá la tabla y el modelo para la clase `Employee`, con la siguiente estructura:
    - `name`: `string` de `150` caracteres, no puede ser nulo.
    - `e_number`: `integer`, no puede ser nulo, debe ser único.
    - `office_id`: `integer`, *foreign key* hacia `offices`.
21. Agregá una asociación entre `Employee` y `Office` acorde a la columna `office_id` que posee la tabla `employees`.
    1. ¿Qué tipo de asociación declaraste en la clase `Employee`?
    2. ¿Y en la clase `Office`?
    3. ¿Qué métodos generó AR en el modelo a partir de esto?
22. Sobre *scopes*:
    1. ¿Qué son los *scopes* de AR? ¿Para qué los utilizarías?
    2. Investigá qué diferencia principal existe entre un método de clase y un scope, cuando se los utiliza para realizar la misma tarea.
    3. Agregá los siguientes scopes al modelo `Employee`:
        - `vacant`: Filtra los empleados para quedarse únicamente con aquellos que no tengan una oficina asignada (o *asociada*).

    - `occupied`: Complemento del anterior, devuelve los empleados que sí tengan una oficina asignada.
4. Agregá este scope al modelo `Office`:
    - `empty`: Devuelve las oficinas que están disponibles (`available == true`) que no tienen empleados asignados.

23. Sobre *scaffold controllers*:

    1. ¿Qué son? Al generarlos, ¿qué operaciones implementan sobre un modelo? ¿Pueden extenderse o modificarse?
    2. ¿Con qué comando se generan?
    3. Utilizando el generator que indicaste en el inciso anterior, generá un controlador de scaffold para el modelo `Office` y otro para el modelo `Employee`.
    4. ¿Qué rutas agregó este generator?
    5. Analizá el código que se generó para los controladores y para las vistas, y luego modificalo para que no permita el borrado de ninguno de los elementos. Enumerá los cambios que debiste hacer para que:
        - Las vistas no muestren la opción de borrar.
        - Los controladores no tenga la acción `destroy`.
        - Las rutas de borrado dejen de existir en la aplicación.
    6. Modificá la vista de detalle (`show`) de una oficina para que, además de la información de la misma que ya presenta, muestre un listado con los empleados que tenga asociados en el cual cada nombre de empleado sea un *link* a su vista de detalle (`employees#show`).

        > Tip: para armar los links, investigá el uso del helper `link_to` y utilizalo.

ActiveModel (AM)

24. ¿Qué son los validadores de AM? ¿Cuáles son los validadores básicos que provee esta librería?
25. Agregá a los modelos `Office` y `Employee` las validaciones necesarias para hacer que sus atributos cumplan las restricciones definidas para las columnas de la tabla que cada uno representa.
26. Validadores personalizados:

    1. ¿Cómo podés definir uno para tus modelos AR?
    2. Implementá un validador que chequee que un string sea un número telefónico con un formato válido para la Argentina.

3. Agregá el validador que definiste en el punto anterior a tu modelo `Office` para validar el campo `phone_number`.

Internacionalización (i18n) y localización (l10n)

28. ¿A qué hacen referencia estos conceptos?
29. Investigá qué metodos provee la clase `I18n` para realizar la traducción (i18n) de texto y la localización (l10n) de valores.
30. Modificá el controlador `PoliteController` que creaste antes para que utilice traducciones al imprimir el mensaje de saludo.
    1. Agregá las traducciones en el *locale* por defecto (inglés). ¿Dónde está ubicado ese archivo? ¿Qué convención debe seguir el nombre de los archivos de traducciones para que Rails sepa a qué *locale* corresponden?
    2. Agregá un nuevo archivo de traducciones para el idioma español, y en él definí los mismos mensajes de traducción que en el inciso anterior, pero esta vez en español.
    3. Modificá la lógica de este controlador para que cambie el *locale* con que mostrará los mensajes internacionalizados en función del parámetro `lang` que reciba:
       - Si no recibe el parámetro, o el mismo no es un *locale* de los reconocidos por la aplicación (`en` o `es`), tomará por defecto el *default locale* de la aplicación (`I18n.default_locale`). Ejemplos de este caso: se accede a `localhost:3000/`, `localhost:3000/?lang=fr`, o `localhost:3000/?lang=english`.
       - Si recibe un locale válido, debe utilizarse ese *locale* para realizar la internacionalización de mensajes en la respuesta a ese *request*. Esto no debe modificar el locale por defecto de la aplicación. Ejemplos de este caso: se accede a `localhost:3000/?lang=es`, y `localhost:3000/?lang=en`.
31. Modificá los *scaffold controllers* que generaste para que utilicen i18n, tanto en las vistas como en los mensajes flash del controlador.

> Tip: Investigá qué helper provee Rails en las vistas para las traducciones.

ActionPack (AP)

32. ¿Qué son los *callbacks* de controladores de AP? ¿Para qué los utilizarías?
33. Tomando como base la lógica que implementaste en `PoliteController` para permitir que se especifique el *locale* a utilizar en la petición, refactorizá eso para que sea un *callback* de ese controlador.

34. Refactorizá nuevamente el *callback* para que el mismo se ejecute también en el resto de los controladores de tu aplicación (los *scaffold controllers* en este caso), sin repetir el código en cada controlador. ¿Cómo hiciste eso?

Referencias

- Ruby on Rails - https://rubyonrails.org
  - APIs
  - Guías
  - Guía básica de ActiveRecord
  - Migraciones de ActiveRecord
  - Validaciones de ActiveRecord
  - Consultas a la base de datos con ActiveRecord
  - Extensiones de ActiveSupport