TTPS opción Ruby

Práctica 5

En esta práctica incorporaremos conceptos del framework Ruby on Rails, desarrollando las posibilidades que nos ofrece para facilitar el desarrollo ágil de aplicaciones web.

Rails

1. El framework está compuesto de diferentes librerías:

   - ActionMailer
   - ActionPack
   - ActionView
   - ActiveJob
   - ActiveModel
   - ActiveRecord
   - ActiveSupport
   - ActionCable

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

   1. ¿Qué funcion principal cumple?
   2. ¿En qué parte(s) del patrón Model-View-Controller (MVC) encajaría?

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.

3. Siguiendo los pasos que enumeraste en el punto anterior, creá una nueva aplicación Rails llamada `practica_cinco` 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 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.

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_cinco"`.
       ¿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 controller 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?
    3. ¿Qué ruta(s) agregó el generator que usaste antes?
    4. ¿Con qué comando podés consultar todas las rutas definidas en tu app Rails?

ActiveSupport (AS)

1. ¿De qué forma extiende AS las clases básicas de Ruby para incorporar nuevos métodos?
2. Investigá qué métodos agrega AS a las siguientes clases:
    1. `String`
    2. `Array`
    3. `Hash`
    4. `Date`
    5. `Numeric`

3. ¿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. `humanize`
15. `sum`

4. ¿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

5. 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)

1. ¿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?
2. ¿De qué forma *sabe* ActiveRecord qué campos tiene un modelo?
3. ¿Qué metodos (*getters* y *setters*) genera AR para los campos escalares básicos (`integer`, `string`, `text`, `boolean`, `float`)?
4. ¿Qué convención define AR para inferir los nombres de las tablas a partir de las clases Ruby? Citá ejemplos.
5. 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 organización tienen sus nombres de archivo?
   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.
      - `address`: `text`.
      - `available`: `boolean`, por defecto `false`, no puede ser nulo.

6. Creá el modelo `Office` para la tabla `offices` que creaste antes, e implementale el método `#to_s`.

7. 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`.
8. 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?
    4. Modificá el mapeo de rutas de tu aplicación Rails para que al acceder a `/` se vaya al controller definido antes (`polite#salute`).

9. Sobre *scopes*:

    1. ¿Qué son los scopes de AR? ¿Para qué los utilizarías?
    2. Investigá qué diferencia principal existe entre un método estático y un scope.
    3. Agregá los siguientes scopes al modelo `Employee`:
        - `vacant`: Filtra los empleados para quedarse únicamente con aquellos que no tengan una oficina asignada (*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.

10. Sobre *scaffold controllers*:

    1. ¿Qué son? ¿Qué operaciones proveen sobre un modelo?
    2. ¿Con qué comando se generan?
    3. Utilizando el generator 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 te generó para los controllers y para las vistas, y luego modificalo para que no permita el borrado de ninguno de los elementos. ¿Qué cambios debés hacer para que las vistas no muestren la opción, el controller no tenga la acción `destroy` y las rutas de borrado dejen de existir en la aplicación?

ActiveModel (AM)

1. ¿Qué son los validadores de AM?
2. 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.

3. 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)

1. ¿A qué hacen referencia estos conceptos?
2. Investigá qué dos metodos provee la clase `I18n` para realizar la traducción (i18n) y la localización (l10n).
3. Modificá el controller `PoliteController` que creaste antes para que utilice traducciones al imprimir el mensaje de saludo.
4. Modificá los controllers de scaffold que generaste para que utilicen i18n, tanto en las vistas como en los mensajes flash del controller.

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

Referencias

- Ruby on Rails - http://rubyonrails.org
    - APIs
    - Guías
    - Guía básica de ActiveRecord
    - Extensiones de ActiveSupport
    - Rails for Zombies (Un poco desactualizado, pero vale la pena mencionarlo)