TTPS Opción Ruby

Práctica 3: Gemas, Bundler y Sinatra

Antes de comenzar recordá:

- Todos los ejercicios deben tener tests (excepto los de Bundler)
- Usá los códigos de estado y verbos HTTP adecuados para cada operación. ver: http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html

Gemas y Bundler

1 - Creá un directorio e inicializalo con **Bundler**. Analizá si bundler provee un mecanismo de inicialización que no sea creando un archivo Gemfile manualmente.

2 - En un proyecto que utilice bundler, instalá la gema **rubywarrior**. Luego corré el comando rubywarrior y seguí las instrucciones en la pantalla (En caso de tener problemas con el comando anterior, utilizá **bundle exec rubywarrior**).

Para más información ver https://github.com/ryanb/ruby-warrior

3 - Escribí un programa llamado **gh-contributors**, que utilice la gema **Octokit** y permita imprimir en pantalla las contribuciones a un proyecto determinado en Github. Por ejemplo:

```
car@latitud:~ttps $ bundle exec ruby gh-contributions.rb
ttps-ruby/capacitacion-ruby-ttps

Contributions for ttps-ruby/capacitacion-ruby-ttps
* chrodriguez (28)
* patriciomacadden (14)
* juanangelsilva (2)
* F-3r (1)
* ggoral (1)
* rapofran (1)
```

Considerá las siguientes operaciones:

- *Octokit.repo 'user/project' devuelve un objeto con los datos de un repositorio*
  - *Asumiendo el objeto anterior se asigna a la variable repo, entonces repo.rels[:contributors].get.data devuelve un arreglo con cada usuario que contribuyó*

Referencias en https://github.com/octokit/octokit.rb

4 - Analizando el proyecto creado para el ejercicio anterior, analizá:

a - ¿En qué archivo se guardan las versiones de las gemas instaladas por bundler?

b - ¿Qué versión de octokit se utiliza al ejecutar con “bundle exec”?

c - Modificá el Gemfile de tu proyecto para que dependa de octokit 2.4.0 y ejecutá “bundle install”. ¿En qué archivo se refleja el cambio de versión luego de instalar (además del Gemfile)?

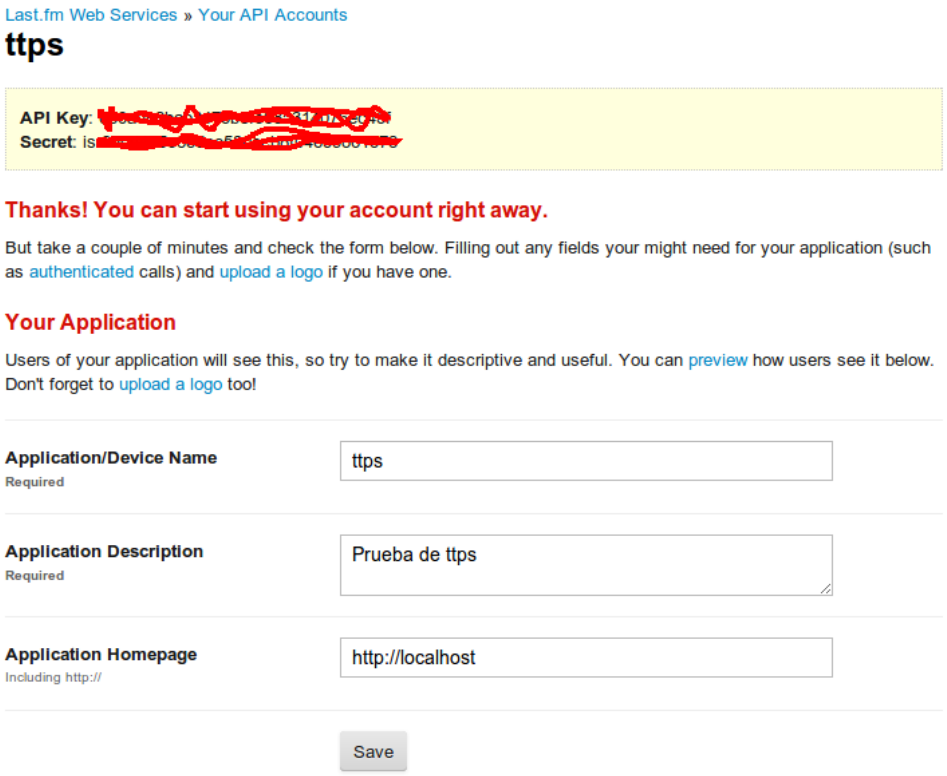
d - Ejecutá “gem search -l octokit” para ver las versiones de octokit instaladas.

e - Ejecutá “bundle install --deployment”, investiguá los contenidos del directorio “vendor” y el directorio “.bundler” creados por este comando. ¿Para qué puede servir ese comando?

f - Leé las recomendaciones de la página de bundler en la sección “Deploying” sobre qué archivos es recomendable versionar y cuales no.

5 - La página que muestra el estado del tiempo actual: http://clima.info.unlp.edu.ar utiliza un servicio similar al empleado por el sitio http://www.info.unlp.edu.ar. Este servicio puede consultarse usando la URL: http://clima.info.unlp.edu.ar/last?lang=es
Usá el comando curl para consultar el servicio

6 - Ingresá en Last.fm: http://www.lastfm.es/ y:
6.1 - Si aún no tenés una cuenta, create una
6.2 - Ingresá luego a: http://www.lastfm.es/api/
6.3 - Creá una API para una aplicación de prueba. Por ejemplo:



6.4 - Tomá nota de tu API key
6.5 - Usá curl para consultar los datos de un artista. Información en http://www.lastfm.es/api/show/album.search

> *Considere que por ejemplo para buscar **iron maiden**, deberá usar como nombre de artista **iron%20maiden***

7 - Usá curl para enviar un GET al servicio http://echo.httpkit.com/. Luego, enviá un POST con los campos nombre y apellido. Probá lo mismo con el método PUT

8 - Escribí una aplicación Sinatra a la que accediendo por un navegador web a la ruta **/saludo** imprima el mensaje *Hola mundo!*. Pero si accede a **/saludo/juan** imprima *Hola juan!*
8.1 - Al iniciar la aplicación, leé los mensajes para entender en qué puerto hay que redireccionar el navegador
8.2 - No te olvides de escribir los tests

9 - Creá una aplicación Sinatra como el siguiente ejemplo e iniciala ya mismo

```
get '/codigo/:code' do
  Integer(params[:code])
end
```

9.1 - Usá Google Chrome y presionar F12 para mostrar las Dev Tools. Esto mostrará una ventanita en la parte inferior del navegador. Asegurate de seleccionar la solapa **Network**
9.2 - Ingresá como URL http://localhost:4567/codigo/404 y verificá qué muestra la columna Status text. Luego probá con diferentes valores como 200, 201, 203, 301, 302, 404, 403, 500
9.3 - Verificá los códigos que quieras según http://en.wikipedia.org/wiki/List_of_HTTP_status_codes

10 - Reescribí la aplicación Sinatra del ejercicio 8, pero ahora, en vez de retornar un string, usá una vista empleando ERB. El template que usarás, deberá incluir una foto y definir una hoja de estilos que cambie el color de fondo a rojo ruby!! La variable que seteará el controlador, deberá usarse por la vista para imprimir el mensaje adecuado. No solo cambia el mensaje sino el título de la página.

11 - Escribí una aplicación Sinatra que tenga 3 rutas:

- /uno
- /dos
- /tres

Las tres acciones compartirán la misma vista:

```
<html>
  <body>
    Action called: <%= @action_name %> at <%=@now %>!
  </body>
</html>
```

11.1 - Implementá la solución de forma tal que la variable **@now**, sólo debe setearse una vez usando filtros.
11.2 - Implementá la solución, pero en vez de usar la variable **@now** usá un helper llamado **now**.

12.1 - Definí un helper de nombre form que reciba los argumentos method y action y devuelva el código HTML de un formulario. Para simular métodos que no sean ni GET ni POST el helper deberá generar un form con método post y agregar un campo oculto con el verbo HTML que realmente queremos utilizar.

Ejemplo: si escribimos en un .erb:

<%= form method: :put, action: '/number' %>

Nuestro helper deberá retornar el string:

<form method="post" action='/number'>

```
  <input type="hidden" name="_method" value="put" />
</form>
```

*Nota: investigá la opción **method_override**.*

12.2 - Agregá el helper submit que dado el texto a mostrar deberá generar un botón submit.

Ejemplo: si agregamos en el template:

```
<%= submit “Enviar” %>
```

Deberá retornar:

```
<input type="submit" value="Enviar">
```

12.3 - El form anterior no es demasiado útil ya que no es sencillo agregar inputs dentro del formulario. Agregale a form la capacidad de recibir bloques, en caso que invoquemos a form con un bloque deberá ejecutarlo para generar todos los inputs que le pidamos.

Ejemplo: si agregamos una ruta “/login”:

```
get '/login' do
  form method: :post, action: '/login' { submit 'Entrar' }
end
```

En el cliente deberá recibir:

```
<form method="post" action="/login">
  <input type="submit" value="Entrar">
</form>
```

Nota: La indentación está sólo por claridad, no es necesario generar el código de salida indentado.

13 - Armar un examen multiple-choice con 5 preguntas de Ruby e implementar una aplicación Sinatra que permita tomar ese examen. El funcionamiento de la aplicación deberá ser el siguiente:
13.1 - **/** y **/login** permiten a un alumno iniciar una sesión escribiendo su nombre y luego redirigen a la primer pregunta.
13.2 - **/question/:number** muestra la pregunta al alumno, un botón retroceder y un botón de avanzar.
13.3 - **/califications** muestra el número de intentos, la calificación de cada intento y la calificación total.

Observaciones:

- Utilizar la cookie session para persistir los datos.
- Para acceder a **/question/:number** y a **/califications** es necesario estar logeado, caso contrario redirigir a **/login**.
- En todas las ventanas deberá haber un link que permita desloguearse.
- En cada sesión un usuario puede rendir hasta 3 veces, luego deberá mostrarse un mensaje indicando que no se puede volver a rendir el examen cuando el usuario intente acceder a **/question/:number**.
- La nota se calcula como el promedio de todos los intentos penalizando con un 10% en cada intento extra.

14 - Implementá un acortador de URLs. Lo que vamos a hacer con un acortador de urls es permitir a los usuarios que a partir de una URL extensa, como por ejemplo http://en.wikipedia.org/wiki/Representational_state_transfer#RESTful_web_APIs, puedan compartir un enlace con una url más corta, por ejemplo: http://example.com/REST. Una vez que se accede a esa URL, el acortador debe redirigir al usuario a la URL completa. La implementación del acortador de url deberá permitir:

* Que el usuario elija la URL acortada de máximo 8 dígitos
* Que el acortador decida la URL acortada de máximo 8 dígitos (por ejemplo usando un hash a partir de la url)
* Que la url generada sea única

Luego, el usuario deberá poder consultar la cantidad de veces que se usó esa URL acortada.

15 - Implementá un servicio web en Sinatra que permita administrar el stock de productos del ejecicio 12 de la práctica 1. Recordar que un producto tiene los siguientes atributos:

- **id**
- **nombre**
- **precio**

La lista de productos se persistirá en alguna estructura de datos que consideren adecuada utilizando los métodos `::set` y `::settings` (ver http://www.sinatrarb.com/configuration.html)
(Nota: tener en cuenta que estos métodos, como explica la documentación, se utilizan para almacenar parámetros de configuración globales de la aplicación y no como capa de persistencia. Lo utilizaremos de esta manera sólo con fin de focalizar el ejercicio)

Para las respuestas y parámetros del servicio tenés que usar como media type el formato JSON. *Ver http://www.ruby-doc.org/stdlib-2.0/libdoc/json/rdoc/JSON.html, http://www.json.org/*

Los servicios que tenés que implementar son:

- ***/products/*:id**

  Que devuelve el detalle de el producto con dicho id.
  El requerimiento debe ser por GET. Una respuesta satisfactoria devuelve:

```
{
  "product": {
    "id": <% id del producto %>,
    "name": <% nombre del producto %>,
    "price": <% precio del producto %>,
  }
}
```

  Ante un error, como por ejemplo que no exista el producto, deberá devolver un error 404 (Not found)

```
{ "error": "Product does not exists" }
```

- ***/products/:id***

  Actualiza el producto indicado con los valores enviados.
  El requerimiento debe ser **PUT** o **PATCH**. Una respuesta satisfactoria devuelve:

```
{ "link": "<% url del recurso modificado %>" }
```

  Si el producto no existe, devuelve un error 404 (Not found)

```
{ "error": "Product does not exists" }
```

  Si se modifica el nombre y ya existe un producto con dicho nombre devuelve un error **409 (Conflict)**

```
{ "error": "Cannot create already existent product" }
```

- ***/products/search/*:query**

  Devuelve una lista de productos cuyo nombre contenga el string que se envía como criterio de busqueda (:query). Si no se envía un criterio de búsqueda, devuelve todos los productos.

  Ante una respuesta satisfactoria devuelve:

```
{
  “products”: [
     {
       “id”: <% id del producto %>,
       “name”: <% nombre del producto %>,
       "price”: <% precio del producto %>
     },
     {
       “id”: <% id del producto %>,
       “name”: <% nombre del producto %>,
       "price”: <% precio del producto %>
     },

     //(...)
   ]
}
```

- **/products**

Agrega un producto al stock.

El requerimiento debe ser por **POST** con los datos de un producto en formato json (con la misma estructura definida en los puntos anteriores). Para esto el request debe llevar la cabecera:

```
Content-Type: ‘application/json’
```

Una respuesta satisfactoria devuelve un enlace al recurso recién creado:

```
{ “link”: <%url del recurso recién creado%> }
```

Todos los campos del producto son obligatorios, por lo tanto, se debe devolver un error **422 (Unprocessable Entity)** si alguno falta o está vacío.

```
{ “error”: “Cannot create product. Invalid data submitted!.” }
```

Si ya existe un producto con el mismo nombre se da cuando ya existe un producto (mismo nombre), devolviendo el error con estado 409 (Conflict)

```
{ “error”: “Cannot create already existent product” }
```