PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA
DEPARTAMENTO DE CIENCIA DE LA COMPUTACIÓN

IIC2513 – Tecnologías y Aplicaciones Web (II/2015)

Profesor: Raúl Montes

Examen

30 de noviembre de 2015

Ejercicio 1 (32 %): Preguntas conceptuales

Responde las siguientes preguntas:

1. (1 pto) ¿Qué diferencias entre RESTful WS y RPC-Style WS hacen a uno más simple que al otro?
2. (1 pto) ¿Cuál es la diferencia entre llamar directamente a una función en JavaScript o entregarla como argumento a la función `$` de jQuery? ¿Dependiendo de qué se debiera usar una u otra forma?
3. (1 pto) Menciona 3 usos útiles que puede tener Ajax y explica en detalle cómo se implementa uno de ellos (en palabras, no código).
4. (1 pto) Explica una diferencia y una semejanza entre el efecto de usar los métodos `cookies` y `session` en el contexto de Rails.
5. (1 pto) Imagina que tienes dos aplicaciones Rails instaladas en el mismo dominio pero en distinto *path* (ej.: `/app1` y `/app2`). ¿Qué condiciones debieran cumplirse para que, habiendo iniciado sesión en una de ellas, al visitar la otra también se muestre con sesión iniciada?
6. (1 pto) En tu experiencia desarrollando una aplicación Web, comenta sobre dos aspectos de dos **estándares Web** diferentes que consideras te han dificultado tu tarea más de lo que debieran. Plantea qué mejora se les podría agregar para resolver esas dificultades. Explica, pero de manera concisa.

Introducción a ejercicios 2 y 3

Hay una nueva forma de tomar café en el Campus San Joaquín, y se llama **D**ynamic **C**offee **C**ourier (también conocida como DCC). Ellos te permiten ordenar tu café favorito mediante una aplicación Web, indicando el café a pedir y el lugar dentro del campus en donde debe ser entregado (¡sí! ¡te lo van a dejar a donde tú estés!).

¿Y por qué "Dynamic"? Porque el precio de los cafés no es constante en el tiempo, sino que puede variar en cada momento, dependiendo de la cantidad de pedidos, de la cantidad de ingredientes disponibles y de otros factores misteriosos que forman parte de su secreto profesional.

Para comprar, entonces, entras en el menú de cafés, seleccionas el tamaño - que puede ser grande, enorme o gigante -, el café que quieres (solo, latte, mocha, macchiato, etc.), tu nombre y el lugar en donde lo quieres recibir. Haces click en "Ordenar" y verás el recibo de tu orden de ese maravilloso brevaje indicando que pronto estará en tus manos o, mejor aún, cruzando tu paladar.

Por si te lo preguntas, el pago se realiza de manera presencial, cuando recibes tu café. Y si el precio de tu elección cambia entre que elijes tu café y presionas "Ordenar" ("dynamic", recuerda) entonces recibirás un error en lugar del recibo/confirmación.

En DCC están recién partiendo y, aunque tienen gran parte de su aplicación Web ya armada usando, por supuesto, Rails, necesitan algo de ayuda para completarla y hacer algunas mejoras. Aunque no pueden pagarte con dinero, te ofrecieron una dosis ilimitada de café diario mientras sigas siendo estudiante en la universidad. Tú, como buen desarrollador(a) de software, por supuesto, aceptaste.

Ejercicio 2 (34 %): Mejorando el formulario para ordenar

Actualmente DCC cuenta con un formulario de pedidos, que comienza con un `select` para elegir el tamaño, luego tiene el listado de los diferentes productos (cafés), cada uno con un *radio button* para poder seleccionarlo (sí, sólo puedes seleccionar 1 café) y campos para ingresar información adicional acerca del pedido. Puedes ver el código HTML del `select` y la representación de cada producto a continuación:

```
<label for="order_size">Size</label>
<select name="order[size]" id="order_size">
  <option value="grande">Grande</option>
  <option value="enorme">Enorme</option>
  <option value="gigante">Gigante</option>
</select>
```

```
<!-- X es el id del producto -->
<label id="product-X" class="product">
  <input type="radio" value="X" name="order[product]" id="order_product_X">
  <h2>Latte</h2>
  <p>Lorem ipsum describiendo el producto</p>
  <span class="price grande">Grande: 1300</span>
  <span class="price enorme">Enorme: 1500</span>
  <span class="price gigante">Gigante: 1600</span>
</label>
```

La idea es realizar dos mejoras en este formulario:

- (2.5 ptos) Para disminuir la cantidad de información desplegada, en lugar de mostrar los precios de los 3 tamaños disponibles de café, mostrar sólo el que se encuentra seleccionado en el `select`. La idea es, entonces, que tanto al inicio como cada vez que el usuario cambie el valor de ese campo, se vea para cada producto sólo el precio correspondiente al tamaño actualmente seleccionado.
- (3.5 ptos) Como los precios son dinámicos y cambian cada cierto tiempo, la idea es que esta página refresque la información respecto a los precios cada cierto tiempo (1 minuto) de manera automática (sin que el usuario tenga que refrescar la página). Eso sí, ten presente que **no puedes simplemente refrescar la página** de manera automática, pues perderías lo que el usuario ha ingresado hasta el momento.

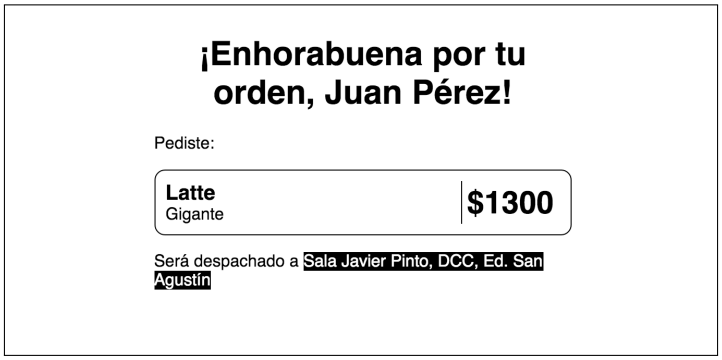
Puedes suponer que ya está creado todo el estado actual de la aplicación (es decir, sin los cambios solicitados anteriormente). Esto incluye la acción del controlador que *renderea* el formulario, la vista del formulario en ERb y un modelo ActiveRecord con los datos `name`, `description`, `price_grande`, `price_enorme` y `price_gigante` (los precios son actualizados en el modelo cuando cambian, así que siempre que cargues el modelo tendrás disponible la última información).

Ahora tú debes escribir **todo** el código necesario adicional (o cambios en código existente) para que se logren los dos objetivos planteados. Especifica el archivo a crear/modificar antes de cada trozo de código que escribas. No te preocupes demasiado del manejo de errores en esta primera versión, puedes simplemente suponer *happy paths*.
Hints:

- cada vez que un `select` cambia de valor, se gatilla el evento DOM `change`
- en JavaScript existen las funciones `setTimeout` y `setInterval`, que reciben una función y un tiempo en ms, y ejecutarán la función luego del tiempo entregado una única vez o en intervalos de ese tiempo, respectivamente.

Ejercicio 3 (34 %): Implementando el recibo

El procesamiento del formulario para crear una orden y la página de recibo no se han implementado aún, así que debes implementarlo tú por completo, desde crear la ruta correspondiente, procesar los datos y mostrar el recibo. DCC sólo tiene el diseño de cómo quiere que se vea el recibo, que puedes ver en la siguiente imagen:



No necesitas hacer nada respecto al formulario. Éste ya está listo (y mejorado en la pregunta anterior) y hará llegar a la aplicación el dato `:order` como uno de los parámetros, que contiene a su vez:

- `:size`: tamaño elegido. Puede ser `:grande`, `:enorme` o `:gigante`.
- `:product_id`: id del producto elegido
- `:name`: nombre del usuario que realiza el pedido
- `:destination`: texto que indica el lugar a donde despachar el producto
- `:total`: valor total a pagar por el café elegido en el tamaño indicado

Con esa información, debes validar primero que el precio a pagar (`:total`) sigue estando vigente. En caso de no seguir vigente (el precio cambió) deberás redirigir al usuario a la ruta `expired_order_path` (no te preocupes por esta página, ya está implementada). En caso de seguir vigente, debes crear la orden (puedes suponer que ya existe el modelo ActiveRecord `Order` con exactamente los mismos atributos que envía el formulario) y mostrar la página de recibo.

Recuerda indicar el archivo que estás creando/modificando antes de cada trozo de código que escribas.

Bonus (0.5 puntos): A DCC le gustaría también poder enviar las órdenes creadas a un servicio de análisis de consumidores mediante una API que ellos proveen. Para ello tan sólo es necesario enviar un *request* `POST` a https://api.canalysis.com/dcc/order con los datos de la orden en formato JSON.

¡Éxito en el examen!