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

IIC2513 – Tecnologías y Aplicaciones Web (II/2016)

Profesor: Raúl Montes

Interrogación 3

8 de noviembre de 2016

Ejercicio 1 (34 %): Preguntas conceptuales

Responde en forma precisa y concisa las siguientes preguntas:

1. (1 pto) Explica cómo funciona el posicionamiento absoluto en CSS. Da un ejemplo utilizando un pequeño trozo de HTML, CSS y/o un dibujo.

2. (1 pto) La siguiente línea de código puede ser perfectamente válida en el contexto de JavaScript (ES5):

```
var amoeba = new Amoeba(name, size);
```

   En muchos lenguajes eso lo interpretamos como la creación de un nuevo objeto a partir de la clase `Amoeba`. Sin embargo, sabemos que JavaScript **no tiene clases**. ¿Qué es lo que está pasando en la línea anterior entonces?

3. (1 pto) Considera los siguientes tres trozos de código:

```
// trozo 1
var foo = 'hello world';
console.log(foo);

// trozo 2
console.log(bar);
var bar;

// trozo 3
console.log(baz);
var baz = 'hello world';
```

   Para cada uno de ellos indica el output que aparecerá en la consola (ya sea output normal o error). Para los trozos 2 y 3 explica además el por qué.

4. (1 pto) Normalmente escribimos código JavaScript en un archivo y hacemos que este archivo se cargue en una determinada página HTML mediante el elemento `script`. En muchas ocasiones, este elemento `script` lo ubicamos dentro del elemento `head`, por lo que el script será descargado e interpretado (ejecutado) **antes** de que se haya cargado el resto del documento (en particular, lo que está dentro del `body`). Eso implica que si queremos que nuestro script interactúe con elementos del documento, éste no podrá hacerlo pues los elementos no están presentes en el momento de ejecución del script. Explica (conceptualmente, no con código) cómo podemos resolver este problema sin recurrir a mover el elemento `script` de posición.

5. (1 pto) En estricto rigor podemos crear una aplicación Web sin utilizar JavaScript. Cada transición en la interacción del usuario con la aplicación puede resultar en un *request* al servidor que entregará como respuesta el nuevo estado que el *browser* mostrará al usuario. Sin embargo, este viaje de ida y vuelta al servidor puede llegar a ser bastante costoso en términos de recursos y tiempo. Menciona un ejemplo en que con JavaScript podemos crear una interacción con el usuario similar a lo que podrías lograr sin JavaScript pero que nos permita evitar el ciclo *request-response*. Explica en palabras (sin código) cómo funcionaría tu ejemplo.

6. (1 pto) La funcionalidad de "olvido de contraseña" usualmente consiste en que ingresas sólo tu correo en la aplicación y ésta te envía un correo en el cual podrás hacer clic en un *link* y, directamente, ingresar una contraseña nueva para tu usuario. ¿Cómo se puede hacer esto de manera segura, sin que cualquier usuario pueda ingresar a una URL cambiando contraseñas de usuarios que no controla? Explica los cambios que necesitarías para lograr esto en una aplicación que cuenta con un sistema básico de usuarios.

Ejercicio 2 (32 %): ¡Qué calor!

La agencia latinoamericana *Detección de Clima y Calentamiento* (DCC) quiere crear el registro más detallado respecto a cómo varía la temperatura de acuerdo a tiempo, posición y condiciones de medición. Para ello ha creado una aplicación Web en que los usuarios registrados pueden ingresar sus propias mediciones de temperatura, indicando datos que permiten luego realizar varios análisis interesantes.

La página de la aplicación en cuestión es un formulario que recopila lo siguiente:

- temperatura, indicando el número y si está en grados celsius o fahrenheit
- posición, indicando latitud, longitud y altura (siempre en metros, así que sólo se necesita pedir 3 números para esto)
- indicación de si la medición fue realizada en interior o a la intemperie
- una entrada de texto libre por si el usuario necesita hacer alguna otra acotación respecto a la medición

Todos estos datos se envían al presionar un botón y luego de ello son almacenados en una base de datos. Si alguno de los datos no se llena o no se hace correctamente (por ejemplo, ingresando un texto en lugar de un número para la temperatura) entonces la página **mostrará un error por la validación asociada** y pedirá al usuario llenarlo correctamente (salvo en el caso de la entrada libre de texto, que es opcional).

Lo que la DCC te pide es que escribas el código asociado únicamente a 2 componentes del MVC, a tu elección, pero utilizando Rails. Por ejemplo, puedes escribir sólo el controlador y el modelo, o sólo la vista y el controlador. Sin embargo, para el componente que elijas no escribir debes explicar en palabras cómo necesita ser implementado para interactuar con el resto de tu solución (de manera que otro desarrollador lo implemente posteriormente). Además, es importante que lo que escribas siga buenas prácticas y recomendaciones en el contexto de Rails, incluyendo ser una solución *resource oriented*.

Nota: No, esta pregunta no necesita de JavaScript. Sólo necesitas manejar bien lo que has puesto en práctica hasta la entrega 4 del proyecto.

Ejercicio 3 (34 %): The next president

En estos momentos está en curso el conteo de votos de una elección presidencial que ha acaparado bastante atención en gran parte del mundo. Dynamic Candidates Choice (DCC) ha estado trabajando en las últimas semanas en un

espectacular sistema de seguimiento en vivo de los resultados de estas elecciones... pero anoche el desarrollador que trabajaba en la interfaz tuvo un arranque de pánico por los posibles resultados de hoy y se fue del país.

Por ello DCC necesita urgente de tu destreza para terminar este sistema y poder lanzarlo mientras aún hay votos que contabilizar. Concretamente, lo que necesitan que termines es lo siguiente:

Parte A (3 puntos)

Considera que ya existe la "clase" `StateResult` con los siguientes métodos:

- `getId` que retorna el id del estado: string sin espacios que identifica a un estado en particular. Ej.: north-carolina
- `getWinner` que entrega el id del candidato con mayor cantidad de votos: ej.: "clinton" o "trump"
- `getCountingPercentage` que entrega el porcentaje de votos escrutados: ej.: 35.4 (refiriéndose a 35.4 %)
- `getElectorsCount` que entrega el número de electores del estado: ej.: 25.

Necesitan que tú escribas la clase `ElectionResult` (en JavaScript, compatible con ES5) cuyo constructor recibe un arreglo de `StateResults` (que no debe quedar público) y que tiene los siguientes métodos:

- `getElectorsResult`: entrega un objeto (o *hash*) en que las *keys* son un id de candidato y los *values* son la cantidad de electores que tiene hasta el momento ese candidato.
- `getWinner`: entrega el id del actual ganador o `undefined` si hay un empate
- `getAverageCountingPercentage`: entrega el promedio simple de los porcentajes de votos escrutados de cada estado
- `getStateResults`: entrega un objecto (hash) de `StateResults` con el id de estado como *key*

Nota: haremos una pequeña simplificación respecto a este proceso. Dentro de un estado, el ganador es simplemente el candidato que tiene más votos. El triunfo en un estado le entrega el voto de todos sus *Electors* a ese candidato. Así, el ganador final de las elecciones no será el candidato que tenga la más alta "votación popular" (votos de cada ciudadano) sino quien tenga la mayor cantidad de votos de electores que representan a cada estado. Ejemplo: consideremos 3 estados, estado1 con 10 electores, estado2 con 15 electores y estado3 con 35 electores. Si candidato1 ganó en estado1 y estado3, y cantidado2 ganó en estado2, tendrán en total 45 y 15 electores cada uno, por lo que gana candidato1. En este mismo ejemplo, puedes notar que en realidad es suficiente para un candidato con ganar en estado3 puesto que tendrá 35 electores, mientras que incluso ganando los otros dos estados el otro candidato tendrá a lo más 25 electores.

Parte B (3 puntos)

Considera la siguiente estructura HTML:

```
<h2>Mi candidato(a) es</h2>
<div>
  <label for="select-clinton">Clinton</label>
  <input type="radio" id="select-clinton" value="clinton">
  <label for="select-trump">Trump</label>
  <input type="radio" id="select-trump" value="trump">
</div>

<h2>Resultados</h2>
```

```
<p id="winner">
  Actual ganador(a): <strong>None</strong>
</p>
<p>
  Porcentaje de escrutinio: <span id="counting-percentage">None</span>
</p>
<div id="states-details">
  <a id="north-carolina">North Carolina</a>
  <!-- más anchors para cada uno de los estados -->
</div>
```

DCC ya tiene toda la obtención periódica de resultados lista. Sólo resta escribir una función que recibirá un objeto `ElectionResult` con toda la información actualizada, y ésta función debe actualizar la interfaz cada vez que sea llamada. Concretamente debe hacer lo siguiente:

- (1.5 puntos) actualizar el ganador y porcentaje de escrutinio donde corresponde. Puedes suponer que existe una constante `CANDIDATES` con el siguiente objeto: `clinton: "Hillary␣Clinton", trump: "Donald␣Trump"`. En caso de empate, ingresa un texto indicándolo en lugar de un nombre de candidato.
- (1 punto) si el ganador es el mismo que el seleccionado en los *radio buttons* superiores, entonces al párrafo con id *winner* se le debe agregar la clase `happy`. De lo contrario, la clase `sad`.
- (0.5 puntos) al hacer clic en cada uno de los *anchors* dentro de `states-details` debes crear, justo después del *anchor*, un div de clase `details` que contenga la cantidad de votos por cada candidato. Al hacerle click a ese div éste debe ser eliminado.

Ayudamemoria jQuery

Las siguientes funciones de jQuery pueden resultarte útiles (listado no exhaustivo ni estrictamente necesario):

- `on`: escucha en los elementos seleccionados el evento indicado en el primer argumento y llama a la función entregada como segundo argumento cada vez que el evento se gatilla.
- `html`: reemplaza el contenido de los elementos seleccionados por lo entregado como argumento (argumento puede ser un objeto jQuery, HTML como string o simple texto).
- `find`: entrega un objeto jQuery asociado a elementos que sean descendientes de los elementos seleccionados y cumplan con el selector entregado como primer argumento.
- `addClass/removeClass`: agrega/elimina la clase entregada como argumento a los elementos seleccionados.
- `val`: entrega el valor (para campos de formulario) del primer elemento seleccionado.
- `replaceWith`: reemplaza enteramente los elementos seleccionados por lo entregado como argumento (argumentos similares a `html`)
- `hide/show`: cambia la propiedad CSS `display` de los elementos seleccionados al valor `none`/anterior
- `remove`: remueve del DOM los elementos seleccionados
- `after/before`: inserta lo entregado como argumento luego/antes de cada elemento seleccionado
- `insertAfter/insertBefore`: inserta los elementos seleccionados luego/antes de cada uno de los entregados como argumento
- `prepend/append`: inserta lo entregado como argumento al principio/final de cada elemento seleccionado