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

IIC2513 – Tecnologías y Aplicaciones Web (II/2016)

Profesor: Raúl Montes

Examen

29 de noviembre de 2016

Ejercicio 1 (25%): Preguntas conceptuales

Responde en forma precisa y concisa las siguientes preguntas:

1. (1 punto) En el contexto de APIs, ¿por qué razones no es recomendable identificar al usuario mediante *cookies* como lo hacemos en el caso de una aplicación Web común?

2. (1 punto) Si sólo tenemos disponible HTML como tecnología Web, únicamente podremos crear una página web estática. Pero con tecnologías en el lado del servidor y con JavaScript (y, a veces, CSS también) en el cliente, podemos tener sitios web dinámicos: el contenido de la página web cambia en respuesta a diferentes contextos o condiciones, entre las cuales puedes considerar también interacción del usuario. Si queremos implementar una cierta funcionalidad dinámica, ¿cómo decidimos si hacer esto con JavaScript/CSS en el cliente, o con la tecnología que estemos usando en el servidor, o con una combinación de ambos? Explica tres criterios que te permitan tomar esta decisión, apoyándote en ejemplos si estimas necesario.

3. (2 puntos) Considera el siguiente código JavaScript, e indica el *output* que se muestra hasta cada marca indicada como comentario. Por ejemplo, en la marca "A", deberás indicar todo lo que se ha mostrado en la consola desde el principio del programa hasta ese comentario (incluyendo la línea marcada), mientras que en la marca "B" deberás indicar todo lo que se ha mostrado en consola posterior al comentario "A" (excluyendo la línea marcada) y hasta llegar al comentario "B" (incluyendo esta línea marcada).

```
var i = 0;
console.log('i = ' + i);  // A
function f() {
  var j = 0;
  i++;
  j++;
  console.log('i = ' + i + ' / ' + 'j = ' + j);
  return function (k) {
    i++;
    j++;
    k++;
    console.log('i = ' + i + ' / ' + 'j = ' + j + ' / ' + 'k = ' + k);
  };
}  // B
var ff = f();  // C
var fff = f();  // D
ff(10);  // E
ff(20);  // F
fff(30);  // G
```

4. (1 punto) Tienes una aplicación para manejar los libros que tienes en tu colección y cuáles de ellos has leído. Una pantalla tiene un listado de libros y un botón "toggle" que al presionarlo hará cambiar un libro de leído a no leído y viceversa. Para hacer efectivo ese cambio es imprescindible un *request* al servidor. Indica qué irá en cada uno de los componentes de ese *request* HTTP (método, path, ...) bajo el supuesto de que sigues una interacción orientada a recursos.

5. (1 punto) Un amigo tuyo te cuenta que está empezando a desarrollar su primera aplicación Web con Rails. ¿Qué buenas prácticas le recomendarías a tu amigo? Menciona y explica 2 que estén relacionadas directamente a Rails y/o desarrollo Web (en lugar de recomendaciones generales respecto al desarrollo de software, como lo sería el utilizar un sistema de control de versiones).

Ejercicio 2 (25 %): Ajax fighting Trojans

Imagínate que tienes un *scaffold* de la entidad `Trojan` recién creado. Escribe (y explica si corresponde) el código de todos los cambios en la aplicación que tendrías que hacer para que la acción de crear un nuevo `Trojan` sea realizada utilizando Ajax. Considera que el comportamiento esperado es que una vez que presionas el botón "Crear" del formulario, éste desaparecerá y en su lugar aparecerá el mensaje "El nuevo Trojan fue creado exitosamente"; en caso de que haya un error al guardar la entidad, se mostrará el mismo formulario inicial con los errores que ocurrieron.

Este formulario no tendrá validación en el lado del cliente (y tampoco se pide en esta pregunta), recuerda que comienzas directamente con lo creado por una generación de *scaffold*. Asegúrate de especificar y escribir los cambios que requieras en todos los frentes para que esto funcione como se especifica (incluidos nuevos elementos que puedas necesitar, o nuevos archivos, o nuevos métodos, etc.). Hay múltiples maneras correctas de lograr este objetivo, así que simplemente elige la que más te acomode.

Ejercicio 3 (25 %): Global Journal

Unos emprendedores tuvieron una ambiciosa (o estúpida, tú juzgas...) idea hace algún tiempo: crear una especie de diario o *journal* en el que todos los usuarios pueden crear entradas en que indican un texto descriptivo y un estado de ánimo (representado por una palabra de un set de opciones). Además, de manera automática, se incluye la ubicación actual ("latitud,longitud") y la fecha/hora actual. Los usuarios pueden ver también un listado de X entradas creadas por otros usuarios, según ciertos criterios internos, y que se van actualizando minuto a minuto. Como no sabían que nombre ponerle a este sistema, se ayudaron con un "web 2.0 buzzwords generator" y llegaron a *Dynamic Collaboration Capable Unified Cloud* (o *DCCUC* como versión corta).

Lo que tienen actualmente es una aplicación cien por ciento web, pero quieren construir las versiones nativas para móbiles. Para ello, determinaron que debían ejecutar lo siguiente:

- Crear una aplicación aparte que proveerá una API JSON basada en tokens (para identificar el usuario) pero que tiene el mismo modelo que la aplicación Web actual.
- Cambiar la aplicación Web actual de manera que en lugar de almacenar y consultar datos desde la base de datos, se comunicará con esta nueva aplicación utilizando su API.

De esta manera, sólo les restará construir las interfaces de usuario de las aplicaciones móbiles, que se comunicarán con la misma API que la aplicación Web. Para llevar esto a cabo, piden tu ayuda para comenzar esta migración. Lo que el DCCUC necesita de ti es lo siguiente:

- (3 puntos) Crear dos métodos en la aplicación API: el que permite crear una entrada de diario y el que permite obtener entradas de diario para listarlas.

- (3 puntos) Cambiar los métodos que crean y listan entradas de diario en la aplicación Web para que, en lugar de usar la base de datos, utilicen esta nueva API.

En ambos casos concéntrate únicamente en los dos métodos del controlador involucrados, y eventuales clases/métodos adicionales que necesites como dependencias de éstos.

Puedes considerar los siguientes supuestos:

- Existe un modelo `User` y la base de datos ya tiene un *token* asignado a cada usuario.
- Existe un modelo `JournalEntry` que representa una entrada del diario, con los atributos `text`, `feeling`, `timestamp`, `position` y `user` (como una relación `belongs_to` con `User`).
- Existe una clase `JournalEntryManager` que tiene un método `get_feed` que recibe un id de usuario y entrega un arreglo de `JournalEntry`s que deben ser mostrados a ese usuario (útil para la acción de listado).
- Se siguen todas las convenciones y buenas prácticas de Rails.
- La autenticación por token puedes hacerla con lo que ofrece Rails por default o algo más elaborado si gustas.
- Cualquier otro supuesto que no contradiga nada del enunciado, y que dejes explicado en tu respuesta.

Ejercicio 4 (25 %): Presidenciables... ?

Determinación de Candidatos por la Comunidad (DCC) publica, semana a semana, un sitio web con los resultados de una encuesta para conocer la opinión de la gente sobre de los eventuales candidatos a la presidencia de nuestro país. Tienen el siguiente código HTML que representa un menú que lista a los posibles candidatos y que tiene datos embebidos sobre los resultados de cada candidato, incluidos mediante el atributo `data-results`:

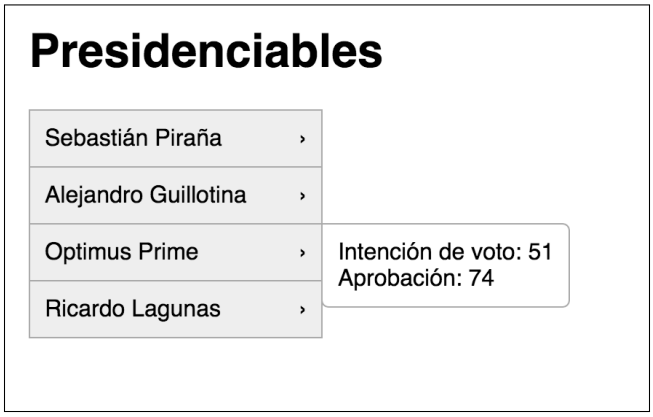
```
<nav id="menu">
  <a data-results="[JSON]">Optimus Prime</a>
</nav>
```

Así como ese elemento `anchor`, hay uno por cada candidato que aparece en la encuesta. El token `[JSON]` que aparece es en realidad un *string* que representa JSON como el siguiente:

```
{
  "voteIntention": 51,
  "approval": {
    "approve": 74,
    "disapprove": 11,
    "nsnr": 15
  }
}
```

Donde `voteIntention` es la intención de voto para ese candidato, `approve/disapprove` es el porcentaje de aprobación/desaprobación que tiene y `nsnr` es el conocido "No sabe/No responde". Recuerda que la función `data` de jQuery te permite obtener esta información directamente como un objeto JavaScript.

Concretamente, lo que DCC necesita de ti es que, **únicamente agregando código CSS y código JavaScript**, implementes que al hacerle click a uno de los candidatos aparezca un cuadro de detalles justo a la derecha del mismo, en que se muestre la intención de voto y la aprobación del candidato. Al hacerle click a cualquier otro candidato, debe aparecer el cuadro de detalles del candidato cliqueado y debe desaparecer el cuadro actualmente abierto, de haberlo. El resultado debe verse como en la siguiente imagen:



No puedes cambiar el HTML que DCC tiene ya hecho. Pero sí puedes modificar el DOM mediante código JavaScript (sí, es cierto, a veces puede no ser lo ideal, pero en este caso no tienes otra opción...).

Bonus: tendrás 0.5 puntos adicionales en este ejercicio si es que mejoras un poco la apariencia de ese cuadro de detalles. Los 3 ítems asociados a la aprobación (approve, disapprove y nsnr) siempre suman 100 %, por lo que puedes representarlos como una barra horizontal que use todo el ancho del cuadro y que tenga una sección verde para aprobación, roja para desaprobación y gris para NSNR, cada una con un tamaño proporcional al número que representan. Dentro de cada barra de color puedes incluir el número asociado, centrado.

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.
- `data`: recibe un nombre de una `key` y entrega el valor de los datos asociados a esa `key`.
- `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.
- `style`: en los elementos seleccionados, asigna estilos de manera *inline* representados por el objeto entregado.
- `replaceWith`: reemplaza los elementos seleccionados por el argumento
- `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 argumentos
- `prepend/append`: inserta el argumento al principio/final de cada elemento seleccionado
- `prependTo/appendTo`: inserta los elementos seleccionados al principio/final de cada elemento entregado como argumento