Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

1 de junio de 2017

IIC2513 – Tecnologías y Aplicaciones Web

Interrogación 3

Instrucciones: Sea preciso: no es necesario escribir extensamente pero sí ser preciso. En caso de ambigüedad, utilice su criterio y explicite los supuestos que considere convenientes. Esta interrogación fue diseñada para durar 120 minutos. Escriba su nombre en cada una de las hojas que entregue.

1. (0.6 pts) Suponga que usted es un usuario registrado de una aplicación web, pero no puede iniciar sesión porque olvidó su contraseña. Al solicitar ayuda con su problema, se le envía un correo con la contraseña que había perdido.

    a. ¿Qué información sobre la aplicación web puede deducir en base al escenario anterior?
    b. Explique por qué esta es una mala política a partir de 2 vulnerabilidades a las que se ven expuestos los usuarios de la aplicación.

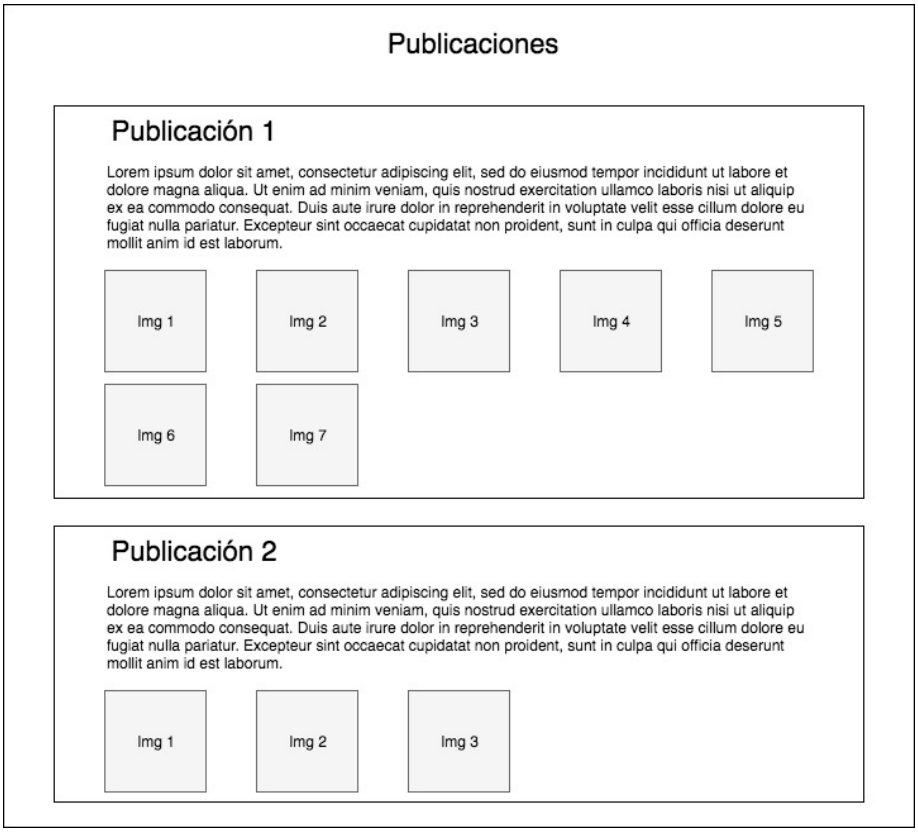
a) Ya que las contraseñas de los usuarios de esta aplicación se pueden obtener en su forma original se puede deducir que no se almacenan en forma de un *hash* procesado.

b)

- Cualquier persona con acceso a la base de datos podría suplantar a los usuarios de la aplicación con sus credenciales.
- La contraseña de un usuario podría enviarse por equivocación a otra persona.
- Muchos usuarios utilizan la misma contraseña para distintas aplicaciones, por lo que si se vulnera la seguridad de esta aplicación se podría acceder a otras aplicaciones suplantando a estos usuarios

2. (0.8 pts) Explique cuáles son los 4 propósitos del *Asset Pipeline* de *Ruby on Rails* y los beneficios de cada uno.
    a. Precompilar: procesa las distintas extensiones de los archivos estáticos para terminar con una sola (e.g. *.js* o *.css*). Permite desarrollar con distintas herramientas, para luego ser procesadas en formatos que soportan los navegadores.
    b. Concatenar: une los distintos archivos *.js* y *.css* de la aplicación en 2 archivos: *application.js* y *application.css*. Disminuye la cantidad de archivos estáticos de la aplicación, lo que reduce las peticiones *http*, por lo que aumenta la velocidad de carga de las páginas de la aplicación.
    c. Minificar: Reduce el tamaño de los archivos *application.js* y *application.css* al quitar elementos innecesarios (e.g. comentarios y espacios). Esto reduce el tamaño de los archivos que se transmiten, lo que aumenta la velocidad de descarga y disminuye la transferencia de datos.
    d. Firmar (*fingerprinting*): agrega un *hash* a los archivos estáticos basados en su contenido. Esto permite aprovechar el *caché* de los navegadores evitando conflictos con las copias que ya no se utilizan.

3. (1.2 pts) Considere la siguiente imagen:



Se le pide que implemente el *HTML* y *CSS* necesarios para lograr ese resultado.

Notas:

- No considere el borde externo de la imagen, no es parte de la página web.
- Puede usar *SCSS* o *CSS*.
- No necesita escribir todo el contenido que se ve en la imagen en su *HTML*. Concéntrese en la estructura y utilice *placeholders*.
- Asigne tamaños y márgenes proporcionales a la representación de los elementos como se ven en la imagen.

HTML:

```
<h1>Publicaciones</h1>
▼<article>
  <h2>Publicación 1</h2>
  ▼<p>
    "Lorem ipsum dolor sit amet, consectetur
    nostrud exercitation ullamco laboris nisi
    eu fugiat nulla pariatur. Excepteur sint
  </p>
  ▼<div>
    <img>
    <img>
    <img>
    <img>
    <img>
    <img>
    <img>
  </div>
</article>
▼<article>
  <h2>Publicación 2</h2>
  ▶<p>…</p>
  ▶<div>…</div>
</article>
```

CSS:

```css
body {
    text-align: center;
    padding: 20px;
}
article {
    width: 90%;
    margin: 20px auto;
    border: 1px solid #000;
    text-align: left;
    padding: 25px 30px;
}
article > div {
    width: 100%;
}
article > div > img {
    width: 100px;
    height: 100px;
    border: 1px solid #000;
    margin: 10px 40px 10px 0;
}
```

4. (0.6 pts)
    a. Al utilizar *jQuery* para manipular el *DOM* se acostumbra que todo el código se encuentre dentro de una función anónima:

    ```
    $(function() {…})
    ```

    Explique qué implica esto.

    b. Por otra parte, al utilizar *Ruby on Rails* con *jQuery* se recomienda reemplazar el código anterior por:

    ```
    $(document).on('turbolinks:load', function() {…})
    ```

    Explique por qué este cambio, detallando qué hace la librería *turbolinks*.

a) Esto implica que todo el código que se encuentre dentro de esa función anónima se ejecutará una vez que el navegador gatille el evento *DOMContentLoaded*. Este evento se gatilla cuando todos los elementos del documento *html* se cargan en el *DOM* del navegador. De esta forma, no ocurrirán errores al ejecutar el código *javascript* si este interactúa con elementos del *DOM* que no se han dibujado aún.

b) La librería *turbolinks* intercepta algunas peticiones del navegador y las realiza a través de *AJAX*. Luego, reemplaza el elemento *body* de la respuesta por el elemento *body* del *DOM* que estaba dibujado. De esta forma, el navegador no gatilla el evento *DOMContentLoaded*, ya que no se cargó una página nuevamente. Por esta razón el código se debe suscribir al evento '*turbolinks:load*', que se gatilla cuando la librería termina de realizar los reemplazos correspondientes.

5. (0.8 pts) Escriba el *output* resultante al ejecutar el siguiente código *JavaScript*:

```
1   var v1 = 0;
2   function f1(v2) {
3     console.log(v1);
4     v1 += v2;
5     var f2 = function () {
6       v1 += v2;
7       console.log(v1);
8     };
9
10    console.log(v1);
11    return function (v3) {
12      v2 += v3 || 0;
13      f2();
14      v1 += v2;
15      console.log(v1);
16    };
17  }
18
19  v1 = 1;
20  var ff = f1(1);
21  ff(1);
22  ff();
```

Respuesta:

1
2
4
6
8
10

6. (1.2 pts) Considere la siguiente estructura *HTML*:

```
<h2>Consultar restricción:</h2>
▼<div>
  <label for="digit-input">Último dígito</label>
  <input type="number" id="digit-input" name="digit-input">
  <span id="error"></span>
</div>
<a id="validate" href="#">Validar</a>
<span id="result"></span>
<h2>Calidad del Aire</h2>
▼<p>
  "Nivel de contaminación: "
  <span id="pollution-index"></span>
</p>
▼<div id="communes">
  <a id="macul" href="#">Macúl</a>
  <a id="la-florida" href="#">La Florida</a>
  <!-- más anchors para cada una de las comunas -->
</div>
```

Asuma que existen 2 constantes:

- `BANNED_DIGITS`: arreglo con *integers* que representan los dígitos con restricción
- `POLLUTION`: *hash* que contiene los niveles de contaminación de cada comuna. Las llaves son el identificador para cada comuna y los valores son *integers* que representan la contaminación correspondiente.

Se le solicita que con la ayuda de *jQuery* implemente las siguientes funcionalidades en *JavaScript*:

- Al hacer *click* en el *anchor validate*:
  - Si el campo '*digit-input*' está vacío, se agrega el texto '*Debe ingresar un número*' al elemento '*error*', con la clase '*red'*. Además, se elimina cualquier texto y clase del elemento '*result*'.
  - Si el campo '*digit-input*' tiene un valor, se elimina cualquier texto y clase del elemento '*error*'. Por otra parte, si el arreglo `BANNED_DIGITS` no contiene elementos, el texto de '*result*' debe decir '*No hay restricción hoy*', y el elemento no debe tener ninguna clase. De lo contrario se verifica si el valor de '*digit-input*' está contenido en el arreglo:
    - De ser así, se reemplaza el texto del elemento '*result'* por '*Su auto tiene restricción*' y se le asigna solamente la clase '*red'*.
    - De lo contrario el texto debe decir '*Su auto no tiene restricción*' y asignar solamente la clase '*green*'.
- Al hacer *click* sobre el *anchor* de una comuna se debe actualizar el texto del elemento '*pollution-index*' por el nivel de contaminación correspondiente a esa comuna.

NOTAS:

- Puede usar la función '`Array.indexOf(value)`', que retorna el índice del primer `value` contenido en `Array`. Si el valor no se encuentra en el arreglo retorna -1.
- Algunas funciones de *jQuery* son:
  - `.on(events,[selector],[data],handler)`
  - `.val([value])`
  - `.text([text])`
  - `.removeClass([className])`
  - `.addClass([className])`
  - `.attr(attributeName,[value])`
  - `.find(selector)`

Respuesta:

```
BANNED_DIGITS = [1, 2, 3];
POLLUTION = {
  macul: 30,
  'la-florida': 25,
};

$(function () {
  $('#validate').on('click', function () {
    var $digitInput = $('#digit-input');
    var lastDigit = $digitInput.val();
    var $message = $('#result');
    var $error = $('#error');
    if (lastDigit) {
      $error.text('');
      $error.removeClass('red');
      if (BANNED_DIGITS.length) {
        if (BANNED_DIGITS.indexOf(+lastDigit) !== -1) {
          $message.text('Su auto tiene restricción');
          $message.removeClass('green');
          $message.addClass('red');
        } else {
          $message.text('Su auto no tiene restricción');
          $message.removeClass('red');
          $message.addClass('green');
        }
      } else {
        $message.text('No hay restricción hoy');
        $message.removeClass('red green');
      }
    } else {
      $error.text('Debe ingresar un número');
      $error.addClass('red');
      $message.text('');
      $message.removeClass('red green');
    }
  });

  $('#communes a').on('click', function () {
    var commune = $(this).attr('id');
    $('#pollution-index').text(POLLUTION[commune]);
  });
});
```

7. (0.8 pts)
    a. Defina y explique qué es *AJAX*, especificando en qué se diferencia de una petición "tradicional".
    b. Explique 3 políticas claramente distintas con las que un servidor puede responder a una petición *AJAX* realizada por un navegador. Describa qué es necesario implementar a nivel de *Backend y Frontend* para cada una.

a) *AJAX* (***A****synchronous* ***J****avaScript* ***a****nd* ***X****ML*) es un conjunto de tecnologías que permiten realizar peticiones *http* que desacoplan el intercambio de datos y la representación de estos. Se diferencian de una petición "tradicional" en que:

- Son asíncronas: la aplicación no se bloquea, un usuario puede seguir interactuando con ella
- No recargan la vista: una vez que se recibe la respuesta el navegador no recarga la vista, sino que el código determina qué se hace con los datos recibidos

b)

1. *Code on demand*: el servidor envía código *javascript* que será ejecutado por el navegador al recibir la respuesta. El *Backend* define el código que se ejecutará y el *Frontend* debe suscribir un evento que ejecutará el código recibido.
2. Vista procesada: el servidor crea un extracto de documento *html* y lo envía como respuesta. Al recibir la respuesta, el código estático de la aplicación que se ejecuta en el navegador inserta este extracto en un elemento predefinido.
3. Datos: el servidor envía datos, y al recibir la respuesta el código que se ejecuta en el navegador determina cómo se utilizarán estos. El *Backend* define los datos que se enviarán y el *Frontend* debe suscribir un evento que cambiará la vista según los datos recibidos.