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

IIC2513 – Tecnologías y Aplicaciones Web (II/2017)

Profesor: Raúl Montes

Interrogación 2

17 de octubre de 2017

Pregunta 1: Conceptos

Responde en forma precisa y concisa las siguientes preguntas:

1. Si más de una regla CSS aplica a un mismo elemento y asignan un valor diferente a una misma propiedad, ¿cómo decide el *browser* con qué valor quedarse? Explica en detalle.

2. Cuando una aplicación Web mantiene una sesión de usuario, ¿dónde se almacenan los datos asociados a esa sesión? Explica en detalle la(s) estrategia(s) normalmente utilizadas.

3. Explica el *box model* de CSS. En ese contexto, explica 3 opciones diferentes que hay para que el contenido de texto simple de dos elementos de tipo bloque, ubicados uno debajo del otro, queden separados entre sí por una distancia de 100 pixeles, indicando una ventaja (o desventaja) de cada opción.

4. En el contexto de manejo de archivos de usuarios en tu aplicación Web, ¿qué implementación elegirías respecto a cómo recibir (upload), almacenar y servir (download) dichos archivos? ¿Cambia en algo tu respuesta dependiendo de si es necesario o no contar con control de acceso para tales archivos?

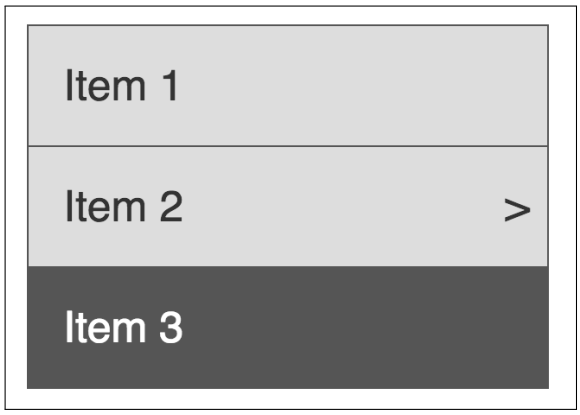
5. En el siguiente trozo de código, la función `doSomething` retorna una promesa que ya sea se resolverá al valor `5` (`Number`) o se rechazará con un `Error` cuya transformación a `String` es `'Ups!'`. Indica el *output* (a raíz de los `console.log`) de este programa tanto para el caso en que la promesa se resuelve como para el caso en que se rechaza.

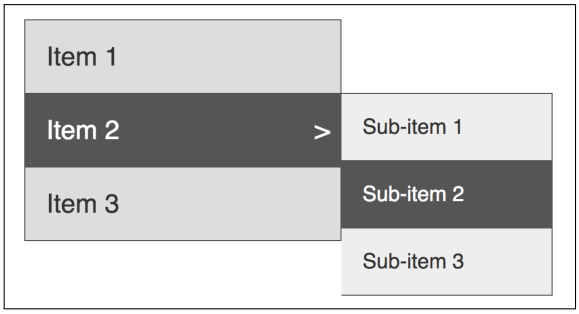
```
doSomething()
  .then((value) => {
    console.log(value);
    return value + 1;
  })
  .catch((reason) => {
    console.log(reason);
    return 10;
  })
  .then((value) => {
    console.log(value);
  });
```

Pregunta 2: a esta interrogación le falta estilo...

En las siguientes imágenes se muestran dos estados de un menú vertical con 3 elementos principales y 3 elementos secundarios asociados al segundo elemento principal. Estos elementos secundarios aparecen cuando el puntero del *mouse* pasa sobre el elemento principal que los contiene.

En la primera imagen el *mouse* está sobre el tercer elemento pricipal (y, por lo tanto, está de color destacado), mientras que en la segunda el *mouse* se posicionó sobre el segundo elemento principal, activando los elementos secundarios asociados, para luego posicionarse sobre el segundo elemento secundario (que aparece también destacado).





Tu tarea en esta pregunta es escribir el código HTML y CSS necesarios para lograr lo que se muestra en las imágenes anteriores. Podrás tener un máximo de 4 (de los 6) puntos en esta pregunta si sólo te encargas del estado de la imagen de la izquierda (es decir, sin considerar sub-menú); podrás acceder a los 6 puntos completos de la pregunta si es que te encargas del estado de la imagen de la derecha únicamente (es decir, suponiendo que el sub-menú está siempre desplegado); y, finalmente, podrás tener un máximo de 7 puntos (1 punto de bonus) si te encargas de implementar ambas imágenes (es decir, con la interacción del *mouse* para desplegar el menú secundario según se explica al principio de esta pregunta).

Respecto a uso de HTML, considera tener un documento limpio, sin abuso de ids ni clases (para así sacar más provecho de selectores), y sacando provecho según se pueda de la semántica de los elementos. Respecto a CSS, puedes usar SCSS si lo deseas. No se aplicará una corrección de compilador en esta pregunta, de manera que no es requisito tener un resultado que un *browser* muestre exactamente como en las imágenes. Pero sí se evaluará según nivel de cercanía a la imágen y que los elementos, atributos, selectores, propiedades y valores sean, cuando menos, muy similares a los correctos.

Pregunta 3: Middlewares

Hemos visto que los *middlewares* son una excelente herramienta para ayudarnos a no repetir código. Con ellos podemos realizar tareas comunes a varios *handlers* de *paths* específicos en nuestra aplicación sin tener que repetirlas en cada uno de ellos.

Uno de los elementos importantes de un *request* HTTP son los *headers*. Ellos permiten que el cliente pueda enviar meta-información sobre la operación y/o entidades involucradas en el *request*.

Tu tarea será escribir y agregar a la aplicación (`app`) un *middleware* de **koa** que revise un *header* del *request* llamado `X-Requested-With` y exponga, para el resto de la cadena de *middlewares*, si es que ese *header* llegó con

valor `XMLHttpRequest`. De esta manera los demás *middleware* podrán, mediante un `if/else` simple, reaccionar de diferente manera si la condición anterior es cierta o no.

Además de escribir el *middleware* escribe y agrega al `router` un *route handler* para un hipotético *path* `/initiatives` que use este valor provisto por el *middleware* que escribiste, a modo de ejemplo (no importa lo que haga, tan sólo que use lo provisto por tu *middleware*).

Tip: Puedes acceder al valor de los *headers* accediendo al objeto `ctx.headers`, que tiene como *keys* los nombres de los *headers* y como *values* los valores de dichos *headers*.

Pregunta 4: Follow the rabbit

Te dieron ganas de programar un juego de acertijos como una aplicación Web. La idea es que tendrás varias páginas diferentes que van presentando diferentes acertijos a los jugadores, y cada uno de estos acertijos te dará pistas sobre cuál es la siguiente URL de la aplicación a la que tienes que acceder.

Si en algún momento accedes a una URL que no existe, se te mostrará un error genérico (como un *status* 404). Si accedes a una URL existente, se te presentará el acertijo correspondiente a esa página. Algunos acertijos pueden llevarte a más de una respuesta correcta, por lo que los caminos de acertijos que sigan distintas personas pueden ser también diferentes.

En algún momento llegarás a una URL final, en la que se termina el juego. Ahí se te mostrará un gran mensaje de felicitaciones y un **listado de las páginas/acertijos (por su URL) por los cuales pasaste** hasta llegar al final del juego.

Quieres hacer esto de manera simple, sin cuentas de usuario, y **sin siquiera necesitar de una base de datos**.

1. Plantea, en palabras, qué solución crearías para implementar este juego utilizando **koa**. No es necesario nada de código. Tan sólo explicar qué elementos usarías, qué función cumplirían, cómo sería el flujo, etc.
2. Como me imagino que quedaste con ganas de implementar algo... escribe el código únicamente de *route handling* de **una** de estas páginas de acertijo y de la página final (la que muestra las felicitaciones y el listado de páginas visitadas). No es necesario que escribas *templates*, sólo necesitas escribir código de *middlewares* (tanto de los *route handlers* como eventuales *middlewares* adicionales si necesitas). Sin embargo, aunque no escribas los *templates*, preocúpate de que tus *middlewares* expongan los datos suficientes para que alguien más pueda escribirlos.

Tip: puedes acceder al *path* del *request* actual utilizando `ctx.path`