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 1

28 de septiembre de 2017

Parte I (50 %): Preguntas teóricas

Responde en forma concisa las siguientes preguntas, usando una hoja diferente para las secciones A y B.

A: Conceptos generales (60 %)

1. (1.5 ptos) En el contexto de HTTP, explica qué son los *headers* y da un ejemplo de alguno indicando además para qué sirve.
2. (1.5 ptos) En el contexto de JavaScript, ¿qué es el *Event Loop*? Explica su funcionamiento.
3. (1.5 ptos) ¿Recuerdas qué significa la C en CSS? Explica en qué consiste ese concepto dentro de esta tecnología.
4. (1.5 ptos) Indica y justifica la veracidad de la siguiente afirmación: el lado del servidor de una aplicación Web sólo recibirá los *requests* que las vistas enviadas al *browser* sean capaces de generar.

B: koa (40 %)

1. (2 ptos) Explica qué es un *middleware* y cómo es el flujo de ejecución de los mismos en koa.
2. (2 ptos) Detalla los pasos necesarios para tener disponible en tu aplicación un modelo `Movie` persistente en la base de datos.
3. (2 ptos) Elige y explica **un** ejemplo, en el contexto de tu proyecto semestral, de cómo el *framework* / *template* utilizado te ha permitido (o te permitiría) evitar repetir código.

Parte II: Preguntas prácticas (50 %)

Las siguientes preguntas podrían requerir que entiendas y/o escribas código. En el caso que sea lo segundo, si bien no será requisito que tu programa esté suficientemente correcto para que al ser "copiado y pegado" funcione a la perfección, sí se espera que sea un indicador de tu conocimiento y capacidad para poder programarlo sin mayor problema (por lo que aunque no lo pasaremos por un "compilador", sí lo vamos a pasar por el filtro "tooMuchPseudocode").

Por favor responde las dos secciones, A y B, en hojas separadas.

Sección A: Promesas, promesas (40 %)

Pregunta 1 (60 %)

Se definen las funciones `getX`, donde `X` puede ser cualquier letra del abecedario. Estas funciones retornan **una promesa** que se resolverá a un cierto valor que llamaremos `valueX`. Si se dice que `Y` depende de `Z` y `W`, implicará que la función `getY` debe recibir como parámetros los valores resueltos de las promesas tanto de `getZ` como de `getW`, en el orden en que se indica. Atención, que estas funcionen deben llamarse con los valores resueltos, no con la promesa por el valor del que dependen.

Para cada uno de los siguientes escenarios, se pide que escribas una función que retorne una promesa por el valor de X, considerando un set de dependencias como las definidas en el párrafo anterior. Debes escribir la función de manera tal que responda lo más rápido posible, "paralelizando" todas las operaciones que puedan ser "paralelizadas". Puedes suponer que estas promesas siempre se resolverán (nunca se rechazarán).

1. (1.2 puntos) X depende de A
2. (1.2 puntos) X depende de A y A depende de B
3. (1.2 puntos) X depende de A y de B
4. (1.2 puntos) X depende de A y de B, A depende de C, B depende de D
5. (1.2 puntos) X depende de A y de B, A depende de C y de D, B sólo depende de C

Pregunta 2 (40 %)

Las funciones `heatWater`, `weightCoffee`, `grindCoffee` y `brewCoffee` son todas funciones asíncronas basadas en *callbacks*. Algunas de ellas, como puedes ver en el siguiente código, requieren recibir otros argumentos además del *callback*. Puedes suponer que todas estas funciones asíncronas llamarán al *callback* recibido con un único argumento, resultado de haber terminado la operación asíncrona.

Luego de entender el código:

1. (3 puntos) Explica lo que sucede en el código anterior cuando llamas a la función `prepareCoffee`.
2. (3 puntos) Si todas las funciones asíncronas en lugar de basarse en *callbacks* ahora retornan promesas, reescribe el código de la función `prepareCoffee` para que ahora también retorne una promesa con el resultado. Tendrás 2 puntos porque tu respuesta sea correcta, y 1 punto adicional si además aprovechas las ventajas de basarse en promesas para que el código quede más simple, con menos código repetido, menos uso de variables.

```
function prepareCoffee(callback) {
  let heatedWater;
  let groundCoffee;
  heatWater((water) => {
    heatedWater = water;
    if (groundCoffee) brewCoffee(heatedWater, groundCoffee, callback);
  });
  weightCoffee((beans) => {
    grindCoffee(beans, (groundBeans) => {
      groundCoffee = groundBeans;
      if (heatedWater) brewCoffee(heatedWater, groundCoffee, callback);
    })
  });
}
```

Sección B: Emisiones de carbono (60 %)

La organización Dioxido de Carbono Controlado (DCC) creó una aplicación móvil para que cualquier persona pueda reportar una emisión de carbono que no se ajuste a las normas. Ahora están desarrollando una aplicación interna para validar estos reportes, basados en un modelo llamado `CarbonEmissionReport`. El requerimiento es muy simple: tendrán una vista que listará todos los reportes de emisiones de carbono que estén en la base de datos y, para cada uno de ellos, habrá un botón que le permitirá al usuario interno "aprobar" ese reporte, lo cual significa que a la instancia de modelo asociada se le cambiará su atributo `approved` al valor `true`.

Un dato importante es que ellos están usando el template del curso, y quieren seguir todas las convenciones y buenas prácticas asociadas.

Pregunta 1 (50 %)

Explica los componentes de la solución que implementarías. No necesitas escribir código, sino sólo explicar dónde y qué cambios sería necesario implementar para lograr esa funcionalidad.

Pregunta 2 (50 %)

Escribe el código necesario explusivamente para la interacción del botón de "aprobar". Específicamente, el trozo del template que muestra el botón de cada reporte, y el código que maneja el *request* gatillado al presionarlo.