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

21 de septiembre de 2017

Parte I (50 %): Preguntas teóricas

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

A: Conceptos (50 %)

1. (2 ptos) En el contexto de HTTP y sus métodos, ¿qué significa que un método sea *idempotente*? ¿qué significa que sea *safe*? Apoya tu explicación con un ejemplo de método que cumpla y que no cumpla cada característica.
2. (2 ptos) Menciona los elementos que tiene un *request HTTP*, indicando para cada uno de ellos el uso o interpretación que se le da al usar HTTP de manera *RESTful* (particularmente, "orientación a recursos").
3. (2 ptos) JavaScript es un lenguaje *single thread*, lo que implica que un programa escrito en este lenguaje se ejecutará en un único proceso y, por lo tanto, nunca habrán dos líneas de código en el programa que se ejecuten simultáneamente (en paralelo). Sin embargo, al escribir un programa en Node.js (con JavaScript) podemos crear un servidor Web, del cual se espera que reciba mútiples *requests* simultáneamente. Significa lo anterior que nuestro programa en Node.js está condenado a procesar un *request* a la vez y que sólo podrá procesar un siguiente *request* una vez haya terminado de procesar el anterior? Explica.

B: koa (50 %)

1. (1.5 ptos) koa únicamente provee una estructura de *middlewares* a la que se le permite procesar los *requests* que llegan y enviar respuestas de vuelta al cliente, por lo que más allá de ello no te fuerza a ningún patrón específico de cómo organizar tu aplicación. ¿Cómo se puede aplicar el patrón MVC en koa? Explica basándote en el *template* que estamos utilizando para el proyecto semestral.
2. (1.5 ptos) Así como en clases realizamos la actividad de detallar todo lo que sucede desde que ingresamos una URL en el *browser* centrándonos en aspectos externos a la aplicación Web misma con la cual podríamos estar interactuando, detalla tú ahora lo que sucede desde el momento en que el *request HTTP* llega al servidor Web y hasta que la respuesta queda lista para ser enviada al cliente, basándonte en la aplicación que has estado construyendo en el proyecto semestral o en la aplicación de ejemplo que vemos en clases.
3. (1.5 ptos) Te pones de acuerdo con un amigo para crear una aplicación en **koa** y, por supuesto, le planteas usar *migraciones*. Sin embargo, tu amigo no está de acuerdo pues considera que no es necesario. ¿Cómo lo convencerías de que es una buena idea usarlas? (considera que tu amigo no es fácil de convencer...)
4. (1.5 ptos) Sin usar nada más que *HTML* y código *backend* (koa), explica cómo puedes lograr tener un elemento *clickeable* que termine siendo procesado por una ruta que procesa método **DELETE** (`router.delete('/path', handler)`). No es necesario que escribas código, sino que expliques todo lo que necesitarías hacer para lograrlo. Pero si prefieres, puedes apoyarte también con 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 ser "copiado y pegado" y funcione a la perfección, sí se espera que sea un fuerte 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").

Responder las dos secciones, A y B, en hojas separadas.

Sección A: Asynchronous JavaScript without XML (40 %)

Pregunta 1 (50 %)

El siguiente trozo de código usa las funciones `prendeParrilla`, `preparaEnsaladas`, `parrilla.asaElPollo` y `parrilla.asaLaCarne`. Todas esas funciones son asíncronas y reciben un *callback* como argumento. Estos *callbacks* son ejecutados y reciben como argumento el valor final de la función una vez que ésta termina su proceso asíncrono. Suponiendo que alguien cambia todas estas funciones para que en lugar de recibir este *callback* retornen una promesa que se resuelva con el valor final del proceso asíncrono, re-escribe este programa de manera que siga funcionando de la misma manera y orden: se preparan cosas "en paralelo" según sea posible pero sólo se comen cuando todas están listas.

Si cumples con que funcione igual pero usando promesas, tendrás 4 puntos. Pero si además lo logras sin usar funciones auxiliares, como `agregaAlPlato` (ni repitiendo código indiscriminadamente en su reemplazo) y sin mantener el arreglo de alimentos (`plato`) de manera global, tendrás 6 puntos.

```
const plato = [];
function agregaAlPlato(comida) {
  plato.push(comida);
  if (plato.length === 3) {
    // come recibe un arreglo de alimentos listos
    come(plato);
  }
}

prendeParrilla((parrillaEncendida) => {
  // asaElPollo entrega un polloAsado asíncronamente
  parrillaEncendida.asaElPollo((polloAsado) => {
    agregaAlPlato(polloAsado);
  });
  // asaLaCarne entrega carneAsada asíncronamente
  parrillaEncendida.asaLaCarne((carneAsada) => {
    agregaAlPlato(carneAsada);
  });
});

// preparaEnsaladas entrega ensalada asíncronamente
preparaEnsaladas((ensaladas) => {
  agregaAlPlato(ensaladas);
});
```

Pregunta 2 (50 %)

En el siguiente trozo de código se usarán diferentes funciones, de las que debes tener presente los siguientes supuestos:

- si el nombre de la función tiene el sufijo `Sync` (ejemplo: `readFileSync`) significa que su código se ejecuta de manera síncrona
- si el nombre de la función tiene el sufijo `Async` (ejemplo: `readFileAsync`) significa que es una función asíncrona y que su valor de retorno será una *promesa* (clase `Promise`).
- todas las funciones, síncronas y asíncronas, en su primera línea de código, escriben en consola el nombre de la función (ejemplo: `console.log('readFileSync');`). Esto se ejecuta de manera síncrona, incluso para funciones asícronas (o sea, antes de comenzar cualquier operación asícrona dentro de la función).
- las funciones síncronas mostrarán en consola `<functionName> finished` justo antes de retornar
- para el caso de las funciones asíncronas, justo antes de que la promesa es resuelta, se mostrará en consola `<functionName> resolved`
- si una función asíncrona recibe una promesa como argumento, debes suponer que esperará la resolución de esa promesa antes de resolver su propia promesa

```
console.log('starting program...');

async function main() {
  console.log('main');
  const waterPromise = heatWaterAsync();
  console.log('heatWaterAsync called');
  const coffee = weightCoffeeSync();
  console.log('weightCoffeeSync called');
  groundCoffeeAsync(coffee)
    .then(async (groundCoffee) => {
      await prepareCoffeeAsync(waterPromise, groundCoffee);
      console.log('coffee prepared');
    });
  console.log('groundCoffeeAsync called');
}
main();
```

Luego de entender el código anterior, responde:

1. Suponiendo que todas las promesas de las funciones asíncronas se demoran exactamente la misma cantidad de tiempo en ser resueltas, ¿cuál sería el *output* en consola del programa anterior?
2. Si supones que la promesa de `heatWaterAsync` se demora mucho tiempo tiempo más que la de `groundCoffeeAsync` y de `prepareCoffeeAsync`, ¿cómo cambiaría la respuesta anterior?

Sección B: (60 %)

Pregunta 1 (60 %)

El **D**epartamento de **C**onteo **C**ósmico - también conocido como DCC - requiere de una aplicación para llevar cuenta de los sistemas planetarios conocidos. Inicialmente esta aplicación debería permitir:

- crear, eliminar, listar y ver un `PlanetarySystem`
- crear un `Planet` asociado a un `PlanetarySystem`
- marcar un `Planet` como `visited`

Indica todas las rutas que la aplicación debería manejar para satisfacer la funcionalidad descrita anteriormente. No tienes que programar estas rutas, pero sí indicar en palabras la información necesaria para definirlas. Deja explícito cualquier supuesto que consideres pertinente.

Pregunta 2 (40 %)

Considerando las rutas determinadas en la pregunta anterior, elige una de ellas y escribe el código necesario. Sólo necesitas escribir el código del `router`, no los *templates* ni modelos. Puedes suponer que todos los modelos necesarios ya existen y que cuentas con los mismos *middlewares* del *template* usado en el proyecto. Si requieres de algún *middleware* adicional para tu respuesta, sí tendrás que escribirlo.

Tendrás el puntaje total (6 puntos) de la pregunta si está correcta, independiente de la ruta elegida. Pero si eliges alguna de las rutas más "interesantes" desde el punto de vista de dificultad, podrás tener hasta 2 puntos adicionales de bonus dependiendo de tu resultado.