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

IIC2513 – Tecnologías y Aplicaciones Web (II/2018)

Profesor: Raúl Montes

Interrogación 3

21 de noviembre de 2018

Pregunta 1 (25%): Conceptos

Responde en forma precisa y concisa las siguientes preguntas (**5 preguntas**, 1.2 puntos cada una):

1. Inicias sesión en una aplicación Web, y luego visitas una página especial de la misma en que se carga una aplicación React. Si la aplicación React realiza intenta luego obtener información desde un *endpoint* que requiere autenticación, ¿funcionará bien? ¿por qué?

2. En el contexto de React, ¿qué es y qué ventaja(s) tiene un componente "puro"?

3. Imagínate que desde código JavaScript corriendo en el browser quieres insertar una enorme lista de títulos de libros como nuevo contenido HTML en la página ya cargada, a través de interacción con el DOM. Los datos de cada libro ya los tienes cargados en una varible. Tienes dos opciones: creas un elemento DOM por cada libro y luego los insertas todos juntos en el DOM existente en la página, o bien creas vas creando cada elemento e inmediatamente insertándolo uno a uno. ¿Son equivalentes ambas opciones? ¿Cuál elegirías y por qué? Si luego te enteras que los datos de los libros no están en memoria sino que hay que obtener cada uno de ellos mediante un request Ajax diferente, indica si tu respuesta anterior cambiaría o no y por qué.

4. En el contexto de manejo de archivos del proyecto semestral, ¿qué ventaja tiene el utilizar "streams" de datos? Explica.

5. En el contexto de una aplicación React existe la posibilidad de configurar "Server Side Rendering" (SSR). Esto significa que en el primer *request* en que se carga la aplicación React, en lugar de entregar un punto de montaje vacío (usualmente algún `div` con un cierto id) el servidor mismo ejecuta y hace *rendering* de la aplicación React similar a cómo si fuera el browser, y el resultado HTML de ejecutar esta aplicación lo inserta dentro del elemento contenedor de la aplicación React. Cuando se envía este HTML (y los assets asociados a la aplicación) al browser, éste igualmente ejecutará la aplicación React como siempre. Explica qué ventajas y/o desventajas tiene el utilizar esta técnica de SSR.

Pregunta 2 (25%): Bee the interpreter

Responde las siguientes preguntas considerando el código indicado en cada una de ellas (1.5 puntos cada una):

1. Explica cuál es la causa más probable de los resultados descritos en los comentarios de este trozo de código (considerando que tanto `buzz` como `collectPollen` son métodos/funciones):

```
const barry = new Bee('Barry');
const janet = new Bee('Janet');
console.log(barry.buzz === janet.buzz); // true
console.log(barry.collectPollen === janet.collectPollen); // false
```

2. De acuerdo al siguiente código HTML y CSS, ¿qué cantidad de pixeles de separación habrá entre ambos textos (nombres)? Explica tu cálculo.

Listing 1: index.html

```
<div id="bees">
  <div>Barry</div>
  <div>Janet</div>
</div>
```

Listing 2: styles.css

```
#bees { padding: 10px; }
#bees div {
  padding-left: 50px;
  padding-bottom: 20px;
  margin: 25px;
  border: 1px solid #bee;
}
```

3. Suponiendo que `fetchBee` es una función que retorna una promesa por una abeja, ¿son equivalentes estos dos trozos en cuánto a la ejecución del programa? Explica.

```
const barry = await fetchBee('Barry');
const janet = await fetchBee('Janet');
console.log(barry.fullname, janet.fullname);
```

```
Promise.all([fetchBee('Barry'), fetchBee('Janet')])
  .then(bees => console.log(bees[0].fullName, bees[1].fullName));
```

4. Al final del siguiente código, indica cómo queda el arreglo `allBees`. ¿Cuántos elementos tiene? ¿Cuántas abejas tiene? Explica los elementos relevantes de este código que llevan al resultado que indicas.

```
function makeHive(maxBees) {
  let count = 0;
  return {
    makeBee() {
      if (count < maxBees) {
        count += 1;
        return new Bee();
      }
      return null;
    },
  };
}
let max = 10;
let hive1 = makeHive(max);
max = 20;
let hive2 = makeHive(max);
max = 40;
let allBees = [];
for (let i = 0; i < 40; i += 1) {
    allBees.push(hive1.makeBee());
    allBees.push(hive2.makeBee());
}
console.log(allBees);
```

Pregunta 3 (25%): Más interactividad en Wican

Hay una (hipotética) sección en Wican que tiene un buscador de iniciativas. Cada búsqueda que un usuario realiza es almacenada en la aplicación, asociada a ese usuario, de manera de poder mostrarlas como sugerencias de búsqueda la siguiente vez que el usuario visite esa sección.

Hemos querido mejorar la interactividad de esta sección transformando esta vista de búsqueda en una página con una aplicación React. Concretamente, el campo de búsqueda, las sugerencias de búsquedas previas y los resultados de búsqueda serán parte de esta aplicación React.

Considerando a Wican como una aplicación Web clásica en este momento, indica todos los cambios necesarios para llevar a cabo este proyecto. No es necesario que escribas código (aunque puedes hacerlo si te resulta más sencillo explicar) pero sí se te pide que expliques en detalle dónde, qué cambios y con qué objetivo necesitarás realizar. Como guía, aquí hay una lista con algunos de los temas (no exhaustiva necesariamente) que deberías abordar:

- setup de la aplicación React en el contexto de la aplicación Web clásica
- qué componentes y con qué función habría que implementar
- comunicación React ¡-¿servidor, para obtener la información necesaria

Pregunta 4 (25%): Reactciona

Wican decidió reinventarse por completo, y esto incluye también el tipo de aplicación Web. Dejará de ser una aplicación Web clásica, y pasará a ser una *single page application* con su *front-end* escrito completamente con React. Esta nueva etapa en la evolución de Wican se llamará *Reactciona*.

Te han pedido algo de ayuda pues el lanzamiento se encuentra muy próximo, y tú consideraste que es una excelente oportunidad para seguir puliendo tus habilidades con este *framework*.

1. (3 puntos) Primero, te pidieron escribir un componente que reciba un arreglo de iniciativas (cada una con atributos `id`, `title` y `url`), y que debe tener como resultado HTML una lista no ordenada, en que cada ítem de la lista tenga el título de la iniciativa y un *link* con el texto "Ir" que lleve a la URL de la iniciativa.
2. (3 puntos) Luego, te piden que escribas otro componente que se encargue de cargar los datos de la iniciativa haciendo un *request* al *path* `/initiatives` – que responde un arreglo JSON con las iniciativas – y, cuando la respuesta llegue, utilizar el componente del punto anterior para mostrarlas.

BONUS: (1 punto) Finalmente, cuando ya pensabas irte, te piden agregar una funcionalidad adicional: que cuando el usuario haga click en una iniciativa se muestre resaltada, dando la ilusión de que está seleccionada. Para ello, basta con mostrar el ítem de lista con la clase `selected`, y el CSS ya incluido hará su trabajo ;)

Nota: Si el punto 2 o el bonus te implica hacer cambios en lo escrito para un punto anterior, puedes evitar reescribir todo siempre y sólo escribir las líneas de código que cambian, siempre y cuando quede muy claro qué estás quitando y/o agregando de contenido anterior.