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

IIC2513 – Tecnologías y Aplicaciones Web (II/2017)

Profesor: Raúl Montes

Examen

23 de noviembre de 2017

Pregunta 1 (25%): Conceptos

Responde en forma precisa y concisa las siguientes preguntas (**6 preguntas**, 1 punto cada una):

1. Te plantean realizar un proyecto de software que se encuentra únicamente a nivel de idea, y lo primero que necesitas hacer es definir si lo mejor es crear una aplicación nativa que no necesite nunca de conexión a internet luego de la instalación/actualización, una aplicación nativa pero que se comunique con una API a través de HTTP o una aplicación Web. Explica 3 criterios para tomar esta decisión y cómo te indican uno u otro tipo de aplicación. Por ejemplo: "Necesidad de X: si es necesario entonces conviene aplicación de tipo Y, sino de tipo Z". **Restricción**: elige criterios tales que para todo tipo de aplicación (los 3 mencionados) exista una evaluación de criterios en que se elige ese tipo.

2. ¿Consideras que la API que implementaste para la entrega 6 es RESTful? Independiente de la respuesta, indica qué restricción REST fue más difícil de (o no lograste) cumplir y qué implementaste (o qué faltó) para cumplir con la misma. Agrega además otras dos restricciones REST que sí hayas logrado cumplir (o que son muy sencillas de cumplir en ese contexto).

3. Durante el semestre tuviste que implementar autenticación dos veces: para la UI Web y para la API. Explica cuáles fueron las diferencias y similitudes entre estas dos implementaciones.

4. JavaScript ha sido **el** lenguaje de programación este semestre. Lo usaste para desarrollar tanto el *backend* como parte del *frontend* de tu aplicación. Explica 2 similitudes y 2 diferencias respecto al uso de este lenguaje (en aspectos como API, forma de programar, configuración, etc.) en el *backend* versus el *frontend*.

5. Imagínate que pudieras volver a algún punto en el pasado de este semestre y te topas contigo mismo. Suponiendo que no causarías ninguna discontinuidad espacio-tiempo que destruya el universo, ¿qué **dos** consejos técnicos importantes (incluyendo argumentos convincentes, claro) te darías? Oriéntalos ya sea a evitar errores cometidos o a obtener una "mejor" aplicación Web como resultado. Deben ser consejos técnicos, así que "dedícale más tiempo", por ejemplo, no cuenta (a pesar de que puede ser un buen consejo en términos generales). Si eres un(a) desarrollador(a) tan seco(a) que no cometiste ningún error o no tienes nada que mejorar, entonces usa esta oportunidad para reforzar en tí mismo(a) algunas de esas geniales decisiones que hayan sido importantes para este buen resultado.

6. Muchas empresas tecnológicas de gran tamaño han hecho una migración desde una aplicación Web monolítica – una única aplicación que se instala y contiene toda la lógica necesaria para responder cualquier *request* – a una aplicación con una arquitectura de microservicios. Esto consiste en que en lugar de una única aplicación (monolítica) se tienen una gran cantidad de aplicaciones muy pequeñas, con una función muy específica, generalmente con su propia fuente de datos, y que se comunican entre sí con una interfaz normalmente REST.

Así, las aplicaciones que reciben *requests* de usuarios generalmente no podrán responder directamente sino que tendrán que comunicarse con estos otros servicios para obtener/actualizar la información que necesitan y así poder generar la respuesta.
Explica 3 ventajas o desventajas que puede tener el enfoque de microservicios respecto a una aplicación monolítica; luego explica e indica cómo resolver 2 dificultades del punto de vista arquitectónico que tendrías que enfrentar si quisieras migrar tu aplicación a microservicios.

Pregunta 2 (25%): Votes DCCounter v2.0

Según lo esperado, tendremos segunda vuelta en la elección presidencial. Y, también según lo esperado, ¡te tocó ser vocal de mesa nuevamente! Esta vez tienes algo más de tiempo así que te propones hacer algo más elaborado para ayudarte en el conteo de votos, y que a la vez no sirva sólo para esta elección sino que para todas las elecciones futuras (pues si saliste elegido como vocal de mesa una vez, probablemente lo sigas siendo en más elecciones :)).

Decides partir creando una API REST, que luego pueda ser consumida por clientes nativos o clientes Web. Esta API debe ser diseñada de tal forma que las aplicaciones cliente puedan proveer la siguiente funcionalidad:

- crear una nueva elección (de manera de poder eventualmente registrar múltiples elecciones)
- agregar o eliminar un candidato a una elección
- actualizar la cantidad de votos de un candidato de una elección
- desplegar la lista de elecciones disponibles y la información detallada de una elección en particular

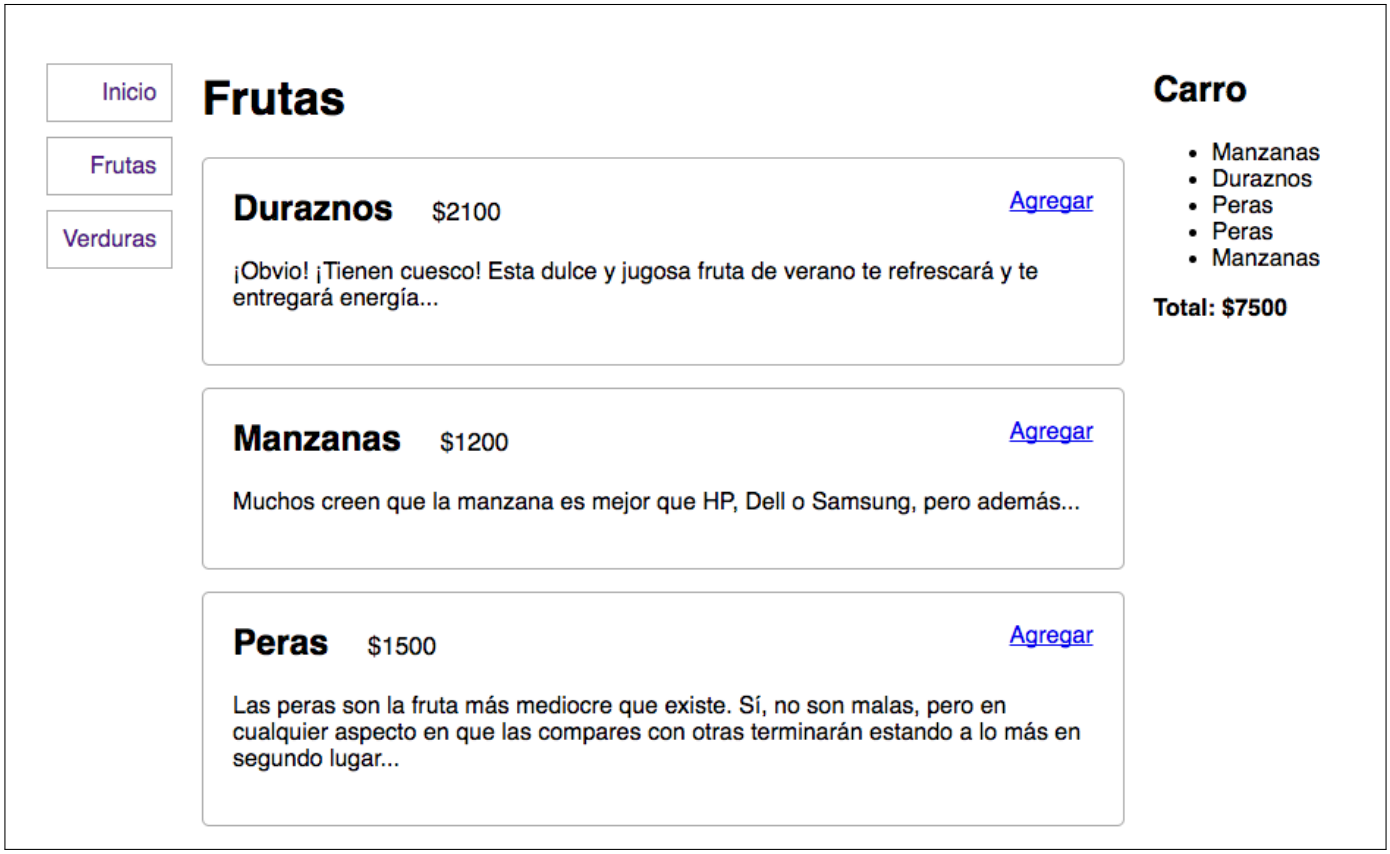
(3 puntos) Escribe la documentación de una API REST que permita realizar lo anterior, indicando los diferentes *requests* soportados con la información necesaria para realizarlos e información importante de las respuestas de los mismos. No es necesario que te alargues demasiado en cada uno; basta con una explicación corta y los detalles relevantes del *request* y *response*.

(1.5 puntos) Ahora elige alguno de los *requests* anteriormente diseñados e impleméntalo en el contexto de una aplicación koa. Puedes suponer existencia de modelos y del resto de *setup* de la aplicación y *middlewares* (indicando si necesitas de alguno en concreto) y concentrarte únicamente en el *route handler*.

(1.5 puntos) Finalmente, pensando en hacer una eventual aplicación Web (*backend*) que consuma esta API, implementa un *route handler* de una aplicación koa que obtenga información del *endpoint* de la API que implementaste y se la entregue a alguna vista para generar un resultado HTML (no es necesario implementar la vista).

Pregunta 3 (25 %): Duraznos con cuesco

Duraznos **c**on **c**uesco (también conocida como DCC) es una nueva frutería que te pidió crear una pequeña aplicación para permitir a sus clientes hacer unos pedidos a través de la Web. Por ahora sólo necesitan parte de su UI, que debería verse como en la siguiente imagen:



Para implementar este trozo de *frontend* tienes dos opciones: o lo haces con puro HTML y CSS, en cuyo caso obtienes un máximo de 4 (de 6) puntos en esta pregunta, o bien lo haces en la forma de un(os) componente(s) React (y, por supuesto, también HTML y CSS), pudiendo optar al máximo de 6 puntos.

Si vas por la ruta de HTML y CSS puedes evitar repetir trozos siempre y cuando expliques lo que se repite y evitar escribir textos largos (como la descripción) si es que se entiende bien dónde iría este elemento en realidad.

Si vas por el camino de React entonces considera que necesitarás también HTML (del componente) y CSS (que puedes considerar como en un archivo aparte) y que el componente principal que debes implementar recibe como `props` un arreglo de objetos fruta en donde cada ítem tiene `name`, `description` y `price`. Además, debes implementar la posibilidad de agregar frutas al carro de compras cuando se hace click en “Agregar”, actualizando los elementos del carro y el total del mismo. No te preocupes por demás funcionalidad como eliminar elementos del carro o no repetir frutas mediante cantidades; basta con que funcione como sugiere la imagen.

Pregunta 4 (25%): Otra vez con los middlewares...

Suponiendo que tienes una aplicación koa en la variable `app`, agrégale un *middleware* que implemente la funcionalidad de *rate limit*. La idea es que no se puedan realizar más de 5 *requests* por minuto desde una misma IP, para así evitar abusos en tu aplicación. Si una IP está realizando el sexto (o superior) *request* a la aplicación, entonces este *middleware* debe hacer que la aplicación responda un error, sin que continúe hacia los demás *middlewares* (y, en particular, *route handlers*).

Hint 1: la clase `Date` permite restar dos objetos obteniendo la diferencia en milisegundos, y tiene métodos como `getHours`, `getMinutes` y `getSeconds`. Crear una instancia sin argumentos creará la representación del "ahora".

Hint 2: `ctx.request.ip` te entrega la IP del cliente que originó el *request*.

Bonus 1: 0.5 puntos adicionales si el estado que necesita tu *middleware* no queda accesible desde el contexto exterior (o sea, sólo la función de middleware puede accederlo y modificarlo).

Bonus 2: 0.5 puntos adicionales si además impides que una misma IP pueda estar ejecutando dos (o más) *requests* simultáneamente en la aplicación (al menos en cuanto a la ejecución posterior a tu *middleware*).