Pontificia Universidad Católica de Chile
Escuela de ingeniería
Departamento de ciencias de la computación
Profesor: Hernán Cabrera

# Entrega 2

## IIC2513 Tecnologías y aplicaciones WEB

**Fecha límite de entrega: viernes 14 de octubre 23:59 hrs.**

Recuerden que los enunciados sólo dan la línea general de la funcionalidad que deben implementar, sin entrar en detalles ni puntos específicos de tal manera que ustedes *demuestren* su capacidad de trabajo en equipo y resolución de problemas.

**Discutan sus supuestos y consideraciones de su implementación y los criterios de evaluación con su ayudante de seguimiento**.

***Salvo excepciones, tendrán total libertad en cuanto a la interfaz, el diseño y la implementación de su trabajo***.

**Fecha límite de entrega, viernes 14 de octubre 23:59 hrs.**

## Indicaciones generales

El objetivo de esta entrega es implementar el concepto de SPA en su aplicación utilizando para ello React y además **comenzar** con la programación de reglas de su aplicación, junto con la creación del modelo de datos y el protocolo (es decir el JSON) que se usará para comunicar la interacción del usuario al servidor y a su vez cómo el servidor comunicará resultados y comportamiento para ser visto por el usuario (cliente).

En esta entrega se espera que como mínimo tengan implementado/elaborado:

1. Mejoras acordadas con su ayudante guía en lo relativo a su front-end
2. Documentación clara y completa de las reglas de negocio de su aplicación[1]. Esta documentación deberá estar online, en su misma aplicación y podrá ser "navegable" por el usuario. Recuerden implementar un "relato" coherente en

[1] Por reglas de negocio se entiende el comportamiento que tendrá su aplicación ante las diferentes acciones que el usuario pueda hacer. Este comportamiento puede ser traducido a efectos en front como también a solicitudes, cambios y respuestas en back.

todo su sitio, mantener estilos, colores, estructura y contenido que hagan entender a un usuario que se encuentran en un “contexto” homogéneo.

3. Comportamiento (acciones) a nivel de servidor de su aplicación. Para probar las reglas de negocio implementadas, el servidor que ustedes programen (en NodeJs, usando Javascript), recibirá las solicitudes (request) usando koa-router y obteniendo el JSON que de cuenta de la interacción (jugadas, simulación, depende de la aplicación) del usuario.

4. Deberán usar HTTP con al menos, para esta entrega, los métodos GET y POST. En su diseño ustedes deberán determinar de manera coherente cuándo corresponde un GET y cuándo corresponde un POST.

   Entendemos que la parte cliente no estará completa y faltará afinar muchos aspectos, por lo tanto esperamos que implementen algunas de las solicitudes en forma completa. Indiquen en su diseño cuáles de las reglas de negocio estarán completas para realizar GET/POST y que a su vez dicho request genere la respuesta requerida (en un response, claro está). Tenga especial atención a los códigos de retorno (200, 204, 404, etc.) y cómo manejarlos de “cara” al usuario y/o a la aplicación de front-end.

5. Entendiendo que hay todavía mucho que definir, diseñar y programar (faltan aún dos entregas para tenerlo todo listo), algunas acciones (request/response) pueden ser simuladas parcial o totalmente. Sin embargo, estas interacciones simuladas deben tener ya la **definición del protocolo/regla de negocio de interacción** que ustedes implementarán. Vale decir, ustedes deberán tener un JSON que defina la acción. Este JSON debe incluir al menos un identificador del usuario (todavía no es exigible la autenticación de usuario) y un conjunto de acciones permitidas según su interacción (por ejemplo, movimientos permitidos de una jugada o acciones válidas en una simulación, según sea el caso)[2].

6. La “respuesta” del Servidor a la interacción será también con un JSON con una estructura también diseñada y documentada por ustedes.

   Es decir, en resumen, el servidor debe poder o bien actuar o bien **simular que recibe acciones,** mediante GET y POST, usando el formato JSON, cuando corresponda y **procesarlas** apropiadamente para luego responder al cliente en un formato JSON también.

[2] Los movimientos/acciones permitidos deberán tener en cuenta el contexto de la interacción, vale decir si por alguna razón una acción realizada es inválida por sus reglas de negocio (por ejemplo en una simulación se están saltando pasos del proceso, ingresando valores incoherentes o en una jugada un movimiento está prohibido o bloqueado) el programa debe retornar como inválida dicha acción e impedirla. De este modo, su servidor actuará como un árbitro que decidirá el resultado de las acciones, entregando los resultados que corresponda, ya sea información de la simulación, resultados, valores, etc. o bien, en el caso del juego, puntos, otorgando victorias, empates y las condiciones y/o estado en los que quedarán los elementos (piezas, personajes, recursos, etc.) de cada jugador en un juego.

El servidor debe **distinguir quién es el usuario** que le está enviando el mensaje JSON, pero no es necesario todavía un protocolo de login seguro.

*NOTA: Recuerden, el servidor solo cumple funciones de **backend**, no expone interfaz de usuario.*

7. Documentación del **modelo de datos** que se utilizará para su base de datos (modelo entidad relación).

8. Implementación del modelo de datos en una **BDD** Postgres. Si lo desea puede implementar el backend en Heroku (o similar) o bien en su PC como localhost. Esa arquitectura debe documentarla y entregar todas las herramientas necesarias para que el ayudante pueda correr su implementación. Si es en una plataforma como Heroku, tal vez entregar credenciales necesarias, si es localhost, entregar las instrucciones para que el ayudante pueda desplegar la solución en su PC..

   NOTA: Pueden usar el ORM **Sequelize** o pueden utilizar "**pg**" (driver ODBC)

9. En el **Readme.md** del repositorio principal (cliente), deberán indicar el enlace a su repositorio de servidor y a su aplicación en Heroku y junto con cualquier información necesaria para su corrección. También deben estipular cualquier cambio o decisión relevante para la corrección.

10. ***Comenzar*** la migración de su aplicación de front a REACT[3]. No todo tiene que ser migrado. Además se entiende que el servidor pueda entregar también páginas estáticas (html puro). Esperamos que las páginas estáticas sean aquellas donde el usuario no tiene ninguna interacción como por ejemplo la página de instrucciones, la página de presentación del equipo o la página de ingreso (o landing page). Es importante que pueda definir con el ayudante los alcances de esta migración.

11. Implementar, en el lado cliente, una página de ingreso a su aplicación que será un archivo index.html, el cual será servido estáticamente por su servidor. Desde este index.html ustedes podrán navegar por el resto de su aplicación. no olviden el .css ni el .js asociados al index.html.

12. Para el caso del lado cliente deberán realizar la entrega de esta versión utilizando todas y cada una de las consideraciones para lado cliente que se entregaron para la entrega anterior (estructura de archivos, archivos mínimos requeridos, etc.)

13. Utilicen desde ya el manual de estilos de programación en JavaScript que les recomendamos seguir (airbnb) https://github.com/airbnb/javascript.

[3] *¿Por qué cambiar a React algo que se pidió originalmente en puro HTML*? Como se explicó en clases, esto obedece a dos razones, la primera una pedagógica en entender cómo realizar páginas estáticas y funcionales y otra corresponde a un tema de diseño, por eso se hizo énfasis en un diseño modular y un uso correcto de barra de navegación, headers, footer y otro código que pudiese ser reutilizado. Si siguieron metodologías adecuadas de diseño, el paso a React será muy fácil

14. Se valorará el uso de Github como elemento de calidad: Uso de branches que se asocien a una funcionalidad específica y commits descriptivos, además de la revisión de PRs.

**La entrega de todos los archivos se hará en el repositorio de Github creado para su grupo**.

***NO se aceptarán***:

- Entregas por mail (ya sea al profesor o ayudantes)
- Entrega en otro sistema que no sea el que se ha provisto para estos efectos (el repositorio Github entregado por el coordinador y deploy con Heroku o equivalente)

## Condiciones y restricciones

1. El framework que se utilizará será Koa.js. Puede usar Express si lo desea
2. Pueden utilizar, si lo desean, Jquery, EJS, SCSS, Flexbox y/o Grid
3. Las reglas de uso y de negocio de su aplicación pueden sufrir variaciones con respecto a la entrega anterior, estas modificaciones **deben señalarlas** en su archivo README.md Las reglas también pueden modificarse, de manera justificada, en las entregas que siguen
4. Se debe usar React, NodeJs y JavaScript

## Recomendaciones

1. Establezcan claramente las reglas de negocio y de uso de su aplicación. Diseñen con cuidado como el servidor interactúa con todos los usuarios. Analicen los casos de uso y el uso concurrente.

2. Recomendamos que separe las reglas por ciertas acciones que deben tener precedencia sobre otras, es decir el **orden de prelación**. Por ejemplo, en la simulación pueden decidir que primero se realicen las acciones de incremento de ciertos insumos sobre otros o en el juego todas las acciones de “sanación” de personajes y luego las acciones de batalla o viceversa. Este tipo de decisiones claramente afectarán el resultado que entregue su aplicación.

3. Aprenderán mucho más si trabajan colaborativamente en su grupo, como equipo en lugar de repartirse el trabajo y realizarlo como unidades independientes.

4. Diseñen muy bien las reglas, el entorno,los turnos, la resolución de empates y conflictos.

5. **Trabajen con tiempo, no esperen a último momento para comenzar con la tarea o despejar dudas.**

6. No traten de resolver aún detalles específicos de integración o comunicación con el servidor. Sin embargo, ya **es momento de que vayan pensando el uso que darán a la base de datos en diferentes casos de uso, como por ejemplo almacenamiento de acciones específicas (simulaciones o jugadas). Por ejemplo, ¿guardarán todas las acciones? solo la última y la penúltima? etc.**

7. Siempre podrán, justificadamente, cambiar alguna regla, mejorar algún aspecto de su aplicación, etc.

8. Recuerden que hay una **evaluación de pares** la cual no es un castigo al que no trabaje sino más bien una protección a los que sí se esfuerzan.

9. Si hay problemas con su compañero(a) y no lo pueden resolver, comuníquese con el profesor o con el coordinador.

## Dudas

Para que todo el curso se vea beneficiado, hagan sus preguntas sobre el material del curso, sobre tecnologías web, y sobre el proyecto a través de los **foros del curso** dispuestos para estos efectos. No se responderá ninguna duda por e-mail.