



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: 13 mayo 23:59 hrs.

Recuerden, los enunciados sólo dan un marco general de la funcionalidad que deben implementar, pero sin entrar en mayores detalles ni puntos específicos de tal manera que ustedes *demuestren* su capacidad de análisis, aplicación de conceptos entregados en clase, trabajo en equipo y resolución de problemas. **No se olviden de acordar los criterios de evaluación con su ayudante de seguimiento.**

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

<b>Fecha límite de entrega, 13 de mayo 23:59 hrs.</b>
---

### Indicaciones

---

El objetivo de esta entrega es comenzar con la ***programación de reglas de su juego en el lado servidor***, junto con la creación del modelo de datos que soportará su juego y el protocolo que se usará para comunicar jugadas y el resultado de éstas. Además, podrá avanzar y mejorar aspectos de usabilidad del lado cliente.

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

1. Documentación clara y completa de las reglas del juego.
2. Modelo de datos mejorado (base de datos)
3. Desarrollo de una API funcional para la comunicación entre cliente y servidor. Esto implica que el servidor recibe peticiones desde el cliente indicando la jugada de un jugador específico y responderá apropiadamente según las reglas del juego establecidas.
4. Comportamiento (acciones) a nivel de servidor de su juego. Para probar las reglas implementadas, el servidor que ustedes programen (NodeJs y Javascript), recibirá (por parámetros o leídas desde un archivo) las jugadas

usando JSON<sup>1</sup> (JavaScript Object Notation), en el protocolo de jugadas establecido por ustedes.

Este JSON puede:

- a. Ser creado a partir de una estructura de datos (objetos, arreglos, etc.) que se debe ingresar y completar en algún archivo .js
  - b. O bien también puede ser un archivo de sistema que el programa leerá desde disco esto es altamente recomendable (para uso de archivos no olviden que existe el módulo “fs”)
  - c. Alguna opción alternativa que usted proponga y documente adecuadamente y que esté acordada por su ayudante.
5. Las acciones, en el lado del servidor deben tener ya la **definición del protocolo de jugada** que ustedes implementarán. Vale decir, ustedes deberán tener un JSON que defina la jugada. Este JSON debe incluir al menos un identificador del jugador (todavía no es exigible la autenticación de usuario) y un conjunto de movimientos permitidos según su jugada.
  6. Los movimientos permitidos deberán tener en cuenta el contexto de la jugada, vale decir si por alguna razón una “pieza” o personaje, no se puede mover y el conjunto de instrucciones dice que sí se puede mover, entonces el programa debe retornar como inválido dicho movimiento e impedirlo. Así pues, **su servidor actuará como un árbitro que decidirá el resultado de los movimientos** entregando puntos, otorgando victorias, empates y las condiciones y/o estado en los que quedarán los elementos (piezas, personajes, recursos, etc.) de cada jugador.
  7. La “respuesta” del Servidor a la jugada se hará siguiendo su protocolo y deberá retornar un código JSON que ustedes deberán **mostrar en el lado servidor** (esto es temporal hasta que tengamos lado cliente. Por ejemplo ahora puede mostrar el JSON completo como texto almacenado en algún objeto que se pueda hacer console.log de él. Para esta entrega no se debe tener una respuesta gráfica en el lado del cliente).

Es decir, en resumen, el servidor debe poder **simular que recibe jugadas** en formato JSON, **procesarlas** apropiadamente y responder en un formato JSON también.

El servidor debe **distinguir quién es el jugador** 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.

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

---

<sup>1</sup> <https://www.json.org/json-en.html>

9. Proponga un modo de simular solicitudes desde el lado cliente (la interfaz) para ejecutar y “finalizar” el turno o jugada, se debe demostrar que se genera un JSON asociado a la jugada y eventualmente se puede “inyectar” al lado servidor y este debe responder a dicha jugada.
10. Implementación del modelo de datos en una **BDD** Postgres.

*NOTA: Aún no es requerido el que utilicen la base de datos, sólo que expongan el modelo de datos que utilizarán. Si quieren investigar, en el futuro cercano veremos el uso del ORM **Sequelize** y del “**pg**” (driver)*

¡Es cierto! Aún no se está comunicando en propiedad (vía HTTP) el cliente con el servidor, pero vayan pensando en eso ya que la siguiente entrega será comunicar ambos mundos.

Cabe aclarar que los **clientes solo reciben la información estrictamente necesaria** para que su jugador (login, puede ser simulado, aunque puede no existir también) pueda informarse del estado del juego y tomar una decisión, según las limitaciones de las reglas del juego.

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

**No basta con que esté el código en Github. Deben entregar en el Readme.md las credenciales y rutas necesarias para que el ayudante pueda conectarse y ejecutar el juego.**

**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 la coordinadora)

## **Condiciones y restricciones**

---

1. Las reglas del juego 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.

## Recomendaciones

---

1. Establezcan claramente las reglas del juego. Diseñen con cuidado como el servidor interactúa con todos los jugadores a la vez. El servidor será el árbitro, pero para que el árbitro dirija, debe haber claridad en todas las reglas y las situaciones que pueden ocurrir.
2. Recomendamos que separe las reglas por ciertas acciones que deben tener precedencia sobre otras. Por ejemplo, pueden decidir que primero se realicen todas las acciones de “sanación” de personajes y luego las acciones de batalla o viceversa. Este tipo de decisiones claramente afectarán el resultado de una jugada.
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 como reserva de jugadas. Por ejemplo, ¿guardarán todas las jugadas? solo la última y la penúltima? etc.**
7. Siempre podrán, justificadamente, cambiar alguna regla, mejorar algún aspecto del juego, etc.
8. Recuerden que se revisará el repositorio para comprobar que todos hayan participado en la programación del proyecto y que se puede penalizar a quienes trabajen poco o mucho menos que el resto del grupo. El trabajo se mide en actividad relevante en github.
9. Si hay problemas con su compañero(a) y no lo pueden resolver, comuníquese con su ayudante de proyecto.

## 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 **del canal de discord del curso o issues del repositorio del curso** dispuestos para estos efectos. No se responderá ninguna duda de tareas por e-mail.