

Entrega #2: UNO Web Edition

Integración con servidor - API REST + Websockets

1. Descripción General

Para dar por culminado el semestre de “Programación Orientada a la Web”, la profesora ha desarrollado un servidor que implementa el juego de UNO para 4 jugadores (1 humano + 3 bots). El cliente web (ustedes) se deberán comunicar con el backend usando HTTP para acciones y WebSocket para actualizaciones en tiempo real.

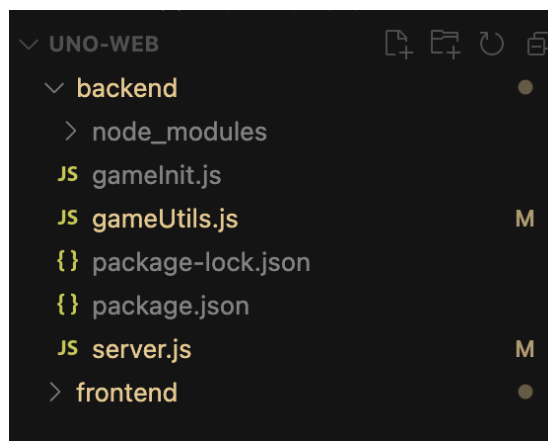
A continuación, podrán clonar el repositorio en cuestión. Si lo desean, pueden crear ramas, siempre identificando el nombre de su grupo.

[REPOSITORIO GITHUB PROYECTO UNO](#)

Árbol del proyecto

El proyecto está conformado por dos directorios base:

- **backend/**: todo el servidor [Node.js](#). Maneja la lógica principal del juego.
- **frontend/**: archivos del cliente (HTML, JS, imágenes de cartas). Es aquí donde deberán ahora añadir su versión del juego, que será adaptada para comunicarse con el servidor.



¿Cómo correr el servidor localmente?

1. Primero, deben tener instalado Node.js (idealmente la versión LTS). Pueden bajarlo de <https://nodejs.org>.
2. Luego, para verificar que este instalado, deben correr el siguiente comando en el terminal (incluyendo npm, que viene por defecto al instalar Node)

```
node -v  
npm -v
```

Si está correctamente instalado, el terminal les arrojará las versiones instaladas.

3. Luego, deberán acceder dentro del terminal de su editor de código y ejecutar el siguiente comando, para entrar al directorio “backend”, donde se encuentra el proyecto de NodeJs.

```
cd backend
```

4. Seguidamente, ejecuten el comando “npm install” o “npm i” para descargar las librerías que necesita el servidor

```
npm install
```

5. Finalmente, para correr el servidor, ejecuten cualquiera de los siguientes comandos para arrancar el servidor que maneja la lógica del juego y la API”

```
npm start
```

o

```
node server.js
```

6. Así, podrán ver que el servidor ya se encuentra corriendo en el puerto 3001:

```
> uno-backend@1.0.0 start  
> node server.js  
  
UNO server listening at http://localhost:3001  
█
```

2. Endpoints HTTP

POST /start : Iniciar una nueva partida (<http://localhost:3001/star>)

- Cuerpo (body): vacío
- Respuesta:

```

{
  finished: false,
  clientCards: [{id: "53", color: "green", type: "number", value: "2"},...],
  discardPile: {id: "16", color: "red", type: "number", value: "8"}
}

```

Annotations for the response:

- `clientCards`: Cartas del usuario real (ustedes)
- `currentColor`: color actual en zona de descarte
- `direction`: sentido del juego (1 para sentido horario, -1 para sentido antihorario)
- `discardPile`: Carta en zona de descarte (última lanzada)
- `finished`: indica si la partida terminó o no
- `gameId`: Identificador de la partida
- `otherPlayers`: indica la sumatoria de puntos de cada jugador (siendo la posición 0 USTEDES y el jugador 1,2,3 respectivamente en sentido horario)
- `turn`: indica el índice del jugador que tiene el turno

Nota: Este endpoint también sirve si quieren reiniciar por completo una partida (pueden añadir un botón en pantalla para esto).

POST /play: Jugar carta (<http://localhost:3001/play>)

- Cuerpo (body):

```

{
  gameId: "c148112d-6f7f-4cc4-a74c-d5864823eb46",
  card: {id: "7", color: "red", type: "number", value: "4"},
  chosenColor: null,
  gameId: "c148112d-6f7f-4cc4-a74c-d5864823eb46"
}

```

Annotations for the request body:

- `card`: carta jugada
- `chosenColor`: color elegido (si se utilizó un comodín de cambio de color, sino, null)
- `gameId`: Identificador del juego

- Respuesta: Estado actualizado del juego (misma respuesta que el Endpoint `/start` pero con datos actualizados)

POST /draw: Robar una carta (<http://localhost:3001/draw>)

- Body:

```

{
  gameId: "c148112d-6f7f-4cc4-a74c-d5864823eb46"
}

```

Annotation for the request body:

- `gameId`: Identificador del juego

- Respuesta: Estado actualizado del juego (misma respuesta que el Endpoint `/start` pero con datos actualizados)

POST /uno: Decir uno (<http://localhost:3001/uno>)

- Body:

```
gameId: "c148112d-6f7f-4cc4-a74c-d5864823eb46" ← Identificador del juego
```

- Respuesta: Estado actualizado del juego (misma respuesta que el Endpoint [/start](#) pero con datos actualizados)

POST /new-round - Iniciar una nueva ronda (no una partida)

(<http://localhost:3001/new-round>)

- Body:

```
gameId: "c148112d-6f7f-4cc4-a74c-d5864823eb46" ← Identificador del juego
```

- Respuesta: Estado actualizado del juego (misma respuesta que el Endpoint [/start](#) pero con **el score acumulado**)

3. WebSocket: Actualizaciones en Tiempo Real

Investiga en la web cómo funcionan los WebSockets y conectate a ([wa://localhost:3001/uno](ws://localhost:3001/uno)). A continuación, te doy una pista de cómo conectarte al servidor vía WebSockets y cómo escuchar los eventos o actualizaciones que lleguen de este servidor:

```
//1. Conectar WebSocket y suscribirse
let ws = new WebSocket('ws://localhost:3001');
// Al enviar el tipo "subscribe", nos suscribimos a una partida en el servidor (enviando también el id de la partida iniciada por http)
ws.onopen = () => ws.send(JSON.stringify({ type: 'subscribe', gameId: 'aquí va el id de la partida' }));

// 2. Escuchar eventos y actualizar UI
ws.onmessage = (e) => {
  let msg = JSON.parse(e.data);
  // en las acciones que lo requieran, vendrá el nuevo estado de juego definido en el servidor
  if (msg.gameState) {}
  // Cada notificación que venga del servidor tendrá un "type" que indicará la acción
  if (msg.type === 'round_score') {}
  if (msg.type === 'new_round') {}
  // etc.
}
```

Lista de eventos WebSocket

Cuando el cliente está suscrito a una partida, recibirá mensajes de los siguientes tipos:

Evento	Descripción	Ejemplo de payload (parcial)
client_play	El jugador humano juega una carta.	{ type: 'client_play', player: 'Player 1', card: {...}, gameState: {...} }
bot_play	Un bot juega una carta.	{ type: 'bot_play', player: 'Player 2', card: {...}, gameState: {...} }
client_draw_from_deck	El jugador humano roba una carta.	{ type: 'client_draw_from_deck', player: 'Player 1', card: {...}, gameState: {...} }
bot_draw_from_deck	Un bot roba una carta.	{ type: 'bot_draw_from_deck', player: 'Player 2', card: {...}, gameState: {...} }
draw_penalty	Un jugador (humano o bot) recibe penalización de cartas.	{ type: 'draw_penalty', affectedPlayer: 2, amount: 2, gameState: {...} }
uno_penalty	Penalización por no decir UNO a tiempo.	{ type: 'uno_penalty', player: 'Player 1', amount: 2, gameState: {...} }
uno_warning	Advertencia: tienes una sola carta, di UNO.	{ type: 'uno_warning', player: 'Player 1', gameState: {...} }
client_uno	El jugador humano dice UNO.	{ type: 'client_uno', player: 'Player 1', gameState: {...} }

bot_uno	Un bot dice UNO.	{ type: 'bot_uno', player: 'Player 2', gameState: {...} }
round_score	Fin de ronda, muestra quién ganó y los puntos sumados.	{ type: ' round_score ', winner: 'Player 2', winnerIdx: 1, roundScore: 120, scores: [0,120,0,0], gameState: {...} } Nota: si gameState.finished el juego ha terminado (ha llegado al score de 500 pts)

Nota: Recuerda que gameState representa el objeto que retorna el endpoint `/start`

4. Indicaciones para la creación del cliente (frontend)

1. Inicia una partida (POST /start) y guarda el gameId.
2. Conéctate por WebSocket y suscríbete con el gameId.
3. Muestra el estado recibido en pantalla (cartas, botón de uno, turno, scores, etc.).
4. Permite jugar cartas (/play) o robar (/draw) según el turno y reglas. Valida desde el lado del cliente que:
 - a. No se pueda seleccionar una carta inválida
 - b. No se pueda seleccionar una carta del mazo si no es tu turno o si tengo al menos una carta válida.
 - c. El botón de uno solo se puede seleccionar si el Jugador 1 tiene una carta.
5. Escucha eventos para actualizar la UI en tiempo real.
 - a. Maneja las notificaciones para mostrar mensajes al usuario.
 - b. Muestra el marcador usando el array scores del estado.
 - c. Muestra una notificación tipo alerta, modal, o como mejor se prefiera, donde se indique quién ha ganado la ronda o la partida si es el caso, y los puntos acumulados.
6. Permite reiniciar la partida si el usuario lo desea. (POST /start)

5. Interfaz final

La interfaz final deberá ser ESTRUCTURALMENTE similar a la imagen siguiente, pero siguiendo los lineamientos de su interfaz actual (o mejorada)

Los elementos a rescatar de la siguiente imagen son:

- Indicador de qué jugador tiene el turno
- Botón de “Reiniciar juego” (llamada)
- Cartas dispuestas en paralelo
- Alerta de notificaciones. Ejemplo:
 - El Jugador 1 jugó X carta
 - El Jugador 2 saltó el turno
 - El Jugador 3 tomó una carta del mazo
 - El Jugador 4 dijo “UNO”
 - Solo te queda una carta, tienes 4 segundos para cantar “UNO”
 - Has sido penalizado por no cantar “UNO”
- Botón de “UNO”, habilitado únicamente cuando las condiciones sean las correctas
- Cartas del Jugador 1 habilitadas según su condición
- Contenedor con el marcador actual de la partida.

Nuevamente, es importante destacar que deben ser creativos en su ejecución y NO dejarse llevar por el diseño de ejemplo, ya que es básico y no muy atractivo a la vista.



Marcador


Player 1: 159

Player 2: 0

Player 3: 0

Player 4: 0


Player 3





1 cards


Restart game


Player 4
























9 cards

Player 2









4 cards

¡Ganaste la ronda!

Puntos obtenidos: 159

Marcador: 159 - 0 - 0 - 0

Continuar

Reiniciar partida

Player 1

