



# Práctica: Who Are You?

Juanan Pereira  
Nagore Barrena

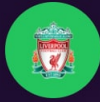


Sistemas Web. 2022-2023

## Who Are Ya? <sup>27</sup>



Awesome


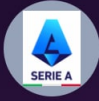

### VIRGIL VAN DIJK



DF

31




### DANILO



DF

31

### MIKEL OYARZABAL



FW

25 ↑

# Índice

<b>1. Introducción y normas del juego</b>	<b>1</b>
1.1. ¿Cómo jugar? . . . . .	1
1.2. Detalles . . . . .	1
1.3. ¿Cuándo cambia el jugador misterioso? . . . . .	1
<b>2. Conseguir perfiles de jugadores</b>	<b>2</b>
<b>3. Conseguir competiciones</b>	<b>3</b>
3.1. Ejercicios . . . . .	4
<b>4. Pobando las llamadas al API de manera sencilla</b>	<b>5</b>
<b>5. Conseguir los equipos de una competición</b>	<b>5</b>
5.1. Añadir nuevos atributos . . . . .	6
5.2. Modificar el nombre de un atributo . . . . .	7
5.3. Modificar los valores de un atributo . . . . .	7
5.4. Ejercicio . . . . .	7
<b>6. Generando el jugador misterioso</b>	<b>7</b>
6.1. Ejercicios . . . . .	8
<b>7. Adivinando jugadores</b>	<b>9</b>
7.1. Ejercicios . . . . .	9
<b>8. ComboBox - Lista de opciones</b>	<b>10</b>
8.1. Ejercicios . . . . .	11
<b>9. Intentos y Game Over</b>	<b>12</b>
9.1. Ejercicios . . . . .	12
<b>10. Estadísticas</b>	<b>13</b>
10.1. Ejercicios . . . . .	14
<b>11. Mejorando la búsqueda del jugador misterioso</b>	<b>14</b>
11.1. Ejercicios . . . . .	15
<b>12. ¿Jugamos otra vez?</b>	<b>15</b>
12.1. Ejercicios . . . . .	15
<b>13. Ejercicios opcionales</b>	<b>16</b>

# 1. Introducción y normas del juego

**Who Are Ya**<sup>1</sup> es un juego que tiene como objetivo adivinar el nombre de un futbolista. Es un juego que se asemeja al famoso 'Quién es Quién', pero basado en ejemplos en vez de en preguntas.

## 1.1. ¿Cómo jugar?

Probando tus conocimientos sobre fútbol debes acertar el futbolista oculto. Para ello se cuenta con un máximo en 8 intentos. El jugador a adivinar juega en alguna liga europea del conjunto "The Big Five" (en adelante, Big5).

Al comenzar el juego se mostrará una imagen difuminada de un jugador. El objetivo es adivinar quién es ese jugador.

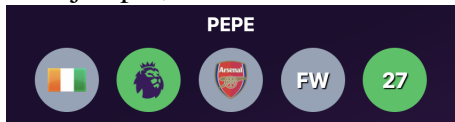
El juego te dará feedback después de cada intento, comparando los datos reales del jugador a adivinar con los datos introducidos por el usuario.

## 1.2. Detalles

Para empezar a jugar, se comienza escribiendo el nombre de un futbolista. Al introducir sólo dos letras del nombre del jugador, se abrirá la lista de jugadores con un nombre que contenga esas letras. Puedes pulsar sobre el nombre de cualquier futbolista para confirmar tu selección. Al hacerlo, los detalles del jugador se cargarán bajo su imagen borrosa (formando una fila de 5 características).

Si has acertado bien alguna de estas características, esta se coloreará en verde. Si no, en gris. Además, si no has adivinado la característica relacionada con la edad del jugador, la aplicación también te dirá si el jugador misterioso es mayor (tiene más edad) o menor.

Por ejemplo, si se introduce el nombre **Pepe** y se obtiene el siguiente resultado:



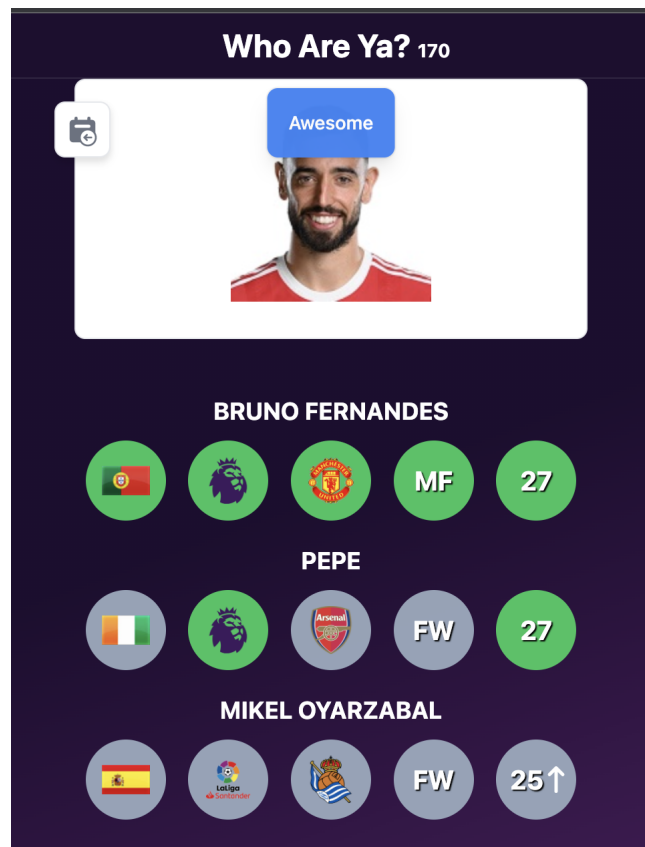
Significara que el jugador misterioso no es de Costa de Marfil, que efectivamente juega en la Premier League, pero no en el Arsenal. El jugador misterioso no es delantero pero sí que tiene 27 años.

## 1.3. ¿Cuándo cambia el jugador misterioso?

El juego se reseteará, eligiendo un nuevo jugador misterioso, cuando el reloj marque las 12am en tu región. Cuando termines una partida (por haber llegado al límite de fallos o bien por haber acertado al jugador) verás la pantalla de estadísticas. En la misma ventana podrás ver también el tiempo que queda para el siguiente juego (ver figura 4).

---

<sup>1</sup><https://missing11.com/who-are-ya/> En realidad debería ser WhoAreYou, pero está escrito en *slang*



En esta práctica recibirás las instrucciones a seguir para programar el juego. Deberás programar y probar las funciones requeridas en cada apartado. Tendrás que subir una versión de tu código a un proyecto (repo) privado de GitHub. Puedes hacer tantos *commit* como quieras pero tendrás que crear un *Git Tag* para cada sección de la práctica cuando así se te indique <sup>a</sup>.

<sup>a</sup>Si nunca has creado un *Git Tag* o si no sabes qué son realmente, no te preocupes, recibirás indicaciones al respecto. Por el momento, puedes comenzar leyendo este documento <https://git-scm.com/book/en/v2/Git-Basics-Tagging>

## 2. Conseguir perfiles de jugadores

A pesar de que el juego whoAreYa utiliza un API de pago, el objetivo de este apartado es aprender a lograr los datos necesarios usando un API gratuito ([football-data.org](https://www.football-data.org/)). Tras finalizar los ejercicios expuestos en los siguientes apartados, usaremos los mismos datos que usa el juego WhoAreYa. De este modo, por un lado demostraremos que sabemos manejar APIs y por otro lado, conseguiremos los datos necesarios para el juego de manera gratuita.

Así, usaremos el sitio web <https://www.football-data.org/> para conseguir los datos de los futbolistas. Esta API gratuita nos ofrece una buena documentación al respecto (<https://www.football-data.org/documentation/quickstart>). Encontraremos la propia API en la siguiente URL: [api.football-data.org](https://api.football-data.org). Nos inscribimos y conseguiremos un Token, que servirá para realizar peticiones HTTP.

### 3. Conseguir competiciones

Usaremos las competiciones de fútbol denominadas "Big5", en concreto:

Premier League ( máxima competición de fútbol en Inglaterra), La Liga ( España), Bundesliga (Alemania). Serie A (Italia) y Ligue 1 (Francia).

Encontraremos todas las competiciones, en forma JSON, en el siguiente enlace: <http://api.football-data.org/v4/competitions>

Dicha respuesta JSON se debe parsear. ¿Cómo? mediante la orden `fetch()`<sup>2</sup>. Puedes usarlo desde el DevTools de Chrome:

```
fetch('http://api.football-data.org/v4/competitions')
  .then(
    r => r.json()
  )
  .then(
    data => {
      console.log(data)
    }
  )
```

#### CORS ¿problemas?

Si recibes problemas CORS en consola, ejecuta DevTools desde la presente página (es decir, incorporate a dicha web y una vez estes ahí abre DevTools y trabaja el comando `fetch()`): <http://api.football-data.org/v4/competitions>

Nos encontraremos que ahí mismo, se especifican datos de 163 competiciones. **Guarda** la respuesta JSON, en un fichero denominado `competitions.json`. Para parsear el fichero JSON usaremos la herramienta `jq` (JSON Query).

#### JQ ¿cómo instalar?

Debes instalar la herramienta `jq`

- En Linux: `apt install jq`
- En macOS: `brew install jq`<sup>a</sup>
- En Windows: `choco install jq`<sup>b</sup>

Para utilizar `jq`, debes abrir un terminal en IntelliJ. Si estas en Windows y el comando, `cat` no funciona (en la terminal PowerShell funciona pero quizás en la terminal cmd no) usa el comando `type`.

<sup>a</sup>si no tienes instalado el paquete `brew` en macOS, este es un buen momento para hacerlo siguiendo las siguientes instrucciones: <https://brew.sh/>

<sup>b</sup>Si no tienes instalado el paquete gestor, este es un buen momento para hacerlo siguiendo las siguientes indicaciones: <https://chocolatey.org/install>

En el fichero JSON guardado, se encuentra en el objeto `data`. Y ahí, el array `competitions`. A continuación, vamos a analizar el ejemplo de un elemento de dicho array. Fíjate que en el atributo `name` se guarda el nombre de la competición o liga.

<sup>2</sup>Ver: <https://developer.mozilla.org/en-US/docs/Web/API/Response/json>

```
$ cat competitions.json | jq -e '.competitions | .[] | select(.id == 2014)'
{
  "id": 2014,
  "area": {
    "id": 2224,
    "name": "Spain",
    "code": "ESP",
    "flag": "https://crests.football-data.org/760.svg"
  },
  "name": "Primera Division",
  "code": "PD",
  "type": "LEAGUE",
  "emblem": "https://crests.football-data.org/PD.png",
  "plan": "TIER_ONE",
  "currentSeason": {
    "id": 1504,
    "startDate": "2022-08-14",
    "endDate": "2023-06-04",
    "currentMatchday": 4,
    "winner": null
  },
  "numberOfAvailableSeasons": 92,
  "lastUpdated": "2022-03-20T09:20:08Z"
}
```

### 3.1. Ejercicios

1. Hemos usado JQ para conseguir el elemento con ID = 2014 del fichero JSON. En los ejercicios que encontrarás a continuación, se pide realizar la misma acción, pero en este caso haciendo uso de JavaScript.

Empecemos con el primer ejercicio: conseguir el archivo JSON con la función *fetch* y a continuación filtrar el objeto con ID=2014. Aclaración, puedes usar el método *filter()*.

2. Nuestro próximo objetivo, será filtrar las ligas Big5. En primer lugar, conseguiremos solo las ligas TIER\_ONE (categoría máxima).

```
$ cat competitions.json | jq -e '.competitions | .[] | select(.plan == "TIER_ONE")'
...
(se deben conseguir 13 resultados).
...

$ cat competitions.json | jq -e ' [.competitions | .[] | select(.plan == "TIER_ONE")] | length '
13
```

Realizar la misma actividad, pero en este caso usando JavaScript (conseguir el archivo JSON mediante el comando *fetch*, y a continuación filtrar las ligas TIER\_ONE.

3. ¿Cómo sería el código para lograr las ligas de España? (usando JS)
4. ¿Cómo sería el código para conseguir el TIER\_ONE de los países ESP, DEU, ENG y FRA? (usando JS)
5. En los ejercicios anteriores, se va a producir un error: en el resultado se ha colado una liga (Championship) que no deseamos (no es de Big5).

```

0: {id: 2016, area: {}, name: 'Championship', code: 'ELC', type: 'LEAGUE', ...}
1: {id: 2021, area: {}, name: 'Premier League', code: 'PL', type: 'LEAGUE', ...}
2: {id: 2015, area: {}, name: 'Ligue 1', code: 'FL1', type: 'LEAGUE', ...}
3: {id: 2002, area: {}, name: 'Bundesliga', code: 'BL1', type: 'LEAGUE', ...}
4: {id: 2019, area: {}, name: 'Serie A', code: 'SA', type: 'LEAGUE', ...}
5: {id: 2014, area: {}, name: 'Primera Division', code: 'PD', type: 'LEAGUE', ...}
length: 6

```

¿Cómo adaptar el comando *fetch* del ejercicio anterior para que no aparezca Championsip en los resultados?

- Utiliza el comando *map()* para conseguir el ID de las ligas resultado del comando previo *fetch*

## 4. Pobando las llamadas al API de manera sencilla

Para comenzar a entender de manera sencilla las llamadas ofrecidas por un API, el IDE IntelliJ ofrece una herramienta especial: *HTTP Client*.

En la aplicación IntelliJ, escogeremos **Tools / HTTP Client / Create Request in HTTP Client**. A continuación, introduciremos los siguientes datos, para conseguir los jugadores del equipo Real Sociedad de LaLiga.

```

GET https://api.football-data.org/v4/teams/90
Accept: application/json
X-Auth-Token: TU_TOKEN

```

### ¡OJO!

Necesitarás un TOKEN para utilizar el API de la web . Para ello deberás registrarte en la mencionada web (es gratuito).

Ahora escoge **Run all requests in file**. El resultado se almacenará en un fichero JSON de manera automática en la carpeta *.idea/httpRequests*. Todas las peticiones HTTP, se guaradan en la carpeta **Scratches and Consoles / Scratches**.

## 5. Conseguir los equipos de una competición

Una vez logrados los ID de las competiciones, podemos conseguir el nombre de los equipos que participan en una competición concreta.

Por ejemplo, sabiendo que el ID de la Premier League es 2021, obtendremos los nombres de los equipos que participan en dicha competición de la siguiente manera (guardar los resultados en un fichero con nombre *premiere.json*):

```

GET https://api.football-data.org/v4/competitions/2021/teams

```

Del mismo modo, podemos obtener los jugadores (*squad*) por cada equipo. Por ejemplo, usando *jq*, para lograr los jugadores del equipo Arsenal:

```
cat premier.json | jq -e '.teams[] | select( .name == "Arsenal FC")'
```

Y sólo quisieramos los nombres de los jugadores:

```
cat premier.json | jq -r '.teams[] | select( .name == "Arsenal FC") | .squad  
  ↪ [] .name'
```

Para conseguir los datos del primer jugador del Arsenal:

```
cat premier.json | jq -r '.teams[] | select( .name == "Arsenal FC") | .squad  
  ↪ [1]'
```

El formato que visualizamos:

```
{  
  "id": 5530,  
  "name": "Aaron Ramsdale",  
  "position": "Goalkeeper",  
  "dateOfBirth": "1998-05-14",  
  "nationality": "England"  
}
```

Pero, el formato que necesitamos para nuestro juego:

```
{  
  "id": 186932,  
  "name": "Mikel Oyarzabal",  
  "birthdate": "1997-04-21",  
  "nationality": "Spain",  
  "leagueId": 140,  
  "teamId": 548,  
  "position": "FW"  
},
```

Como observamos, debemos cambiar los nombres de algunos de los atributos (dateOfBirth:birthDate), introducir unos nuevos (leagueId, teamId) y adaptar el valor de un atributo (Goalkeeper:GK).

## 5.1. Añadir nuevos atributos

Lograr el ID de cada equipo (.id), el ID de la liga (2021 para el caso de la Premier League) y se lo asignamos cada uno de los miembros del equipo

```
cat premier.json | jq -r '.teams[] | .squad[] += { "teamId" : .id, "leagueId":  
  2021}'
```

Y, si sólo necesitamos los datos de los jugadores:

```
cat premier.json | jq -r '[ .teams[] | .squad[] += { "teamId" : .id, "leagueId"  
  ": 2021} | .squad | flatten[] ]'
```



## 5.2. Modificar el nombre de un atributo

Queremos que el atributo `dateOfBirth` sea `birthDate`:

```
cat premier.json | jq -r '.teams[] | .squad[] += { "teamId" : .id, "leagueId": 2021} | .squad | flatten[] | with_entries(if .key == "dateOfBirth" then .key = "birthDate" else . end)'
```

## 5.3. Modificar los valores de un atributo

Además de todo lo anterior, queremos cambiar los valores de los siguientes atributos:

Offence -> FW

GoalKeeper -> GK

Midfield -> MF

Defence -> DF

En el siguiente código sólo se visualizan las modificaciones de Offence y Midfield (los otros dos, siguiendo el patrón, son deducibles y evidentes).

```
cat premier.json | jq -r '.teams[] | .squad[] += { "teamId" : .id, "leagueId": 2021} | .squad | flatten[] | with_entries(if .key == "dateOfBirth" then .key = "birthDate" else . end) | with_entries(if .key == "position" and .value == "Offence" then .value="FW" else if .key == "position" and .value=="Midfield" then .value="MF" else . end end)'
```

## 5.4. Ejercicio

1. Partiendo desde el fichero `premiere.json` original, realiza todos los cambios hechos mediante `jq`, pero con JavaScript.

## 6. Generando el jugador misterioso

Después de haber visto que podemos generar todos los JSON que necesitamos a través de una API gratuita, a partir de este apartado nos centraremos en la programación del juego WhoAreYa.

Descárgate los ficheros `fullplayers.json` y `solution.json`. En el primero de los ficheros (`fullplayers.json`) aparecen todos los datos de los jugadores que juegan en las ligas Big5 (Premiere, Bundesliga, LaLiga, League1, SerieA): identificador del jugador, nombre, fecha de nacimiento, nacionalidad, identificador del equipo, posición en el campo, número del elástico y el identificador de la liga.

En el segundo de los ficheros (`solution.json`), se guardan los identificadores de algunos de los jugadores en un array. Los jugadores que estén en dicho array, serán las soluciones diarias. ¿Cómo se calcula la solución diaria? Indexando el array de la siguiente manera: `arraya[differenceInDays(new Date("08-19-2022"))-1]`.

## 6.1. Ejercicios

1. Programa la función *differenceInDays(base)*. Esta función recogerá un parámetro como entrada (*base*)<sup>3</sup>. Así, si le indicamos el 1 de Marzo de 2022, y si hoy es el 1 de Septiembre de 2022, nos deberá devolver el número 185. Por otro lado, si le pasamos como parametro de entrada el 18 de Agosto y hoy es el 2 de Septiembre del mismo año, nos devolverá el valor 16. Es decir, la función devuelve los días de diferencia entre el día de hoy y el día indicado en parámetro de entrada. En ambos ejemplos también se cuenta el último día.
2. Implementa la función *fetchJSON(file)*. La presente función recoge como parametro de entrada el fichero JSON que se quiere leer, y devolverá el contenido de dicho fichero en un objeto de tipo JSON.
3. Carga *fullplayers.json* y *solution.json*<sup>4</sup>. Implementa la función denominada *getSolution(players, solutionArray, differenceInDays)*. Como se observa esta función recoge tres parametros de entrada (el array JSON de los jugadores, el array JSON de las soluciones, y los días transcurridos entre la fecha del día actual y la fecha 18/08/2022) y devuelve el objeto del jugador correspondiente al día de hoy. Por ejemplo, si hoy es 2 de septiembre, la distancia en días entre fechas será de 16, así al calcular *solutions[16-1]* el resultado es el jugador con ID 159208 (nombre del jugador: Danilo). La función devolverá, por lo tanto, el objeto referido a dicho jugador (el objeto completo en formato JSON).



En el ejercicio anterior, además de devolver el objeto del jugador, sería interesante poder visualizarlo también por consola. De esta manera, los ganadores obtendrá información valiosa para saber si el código está bien programado.

4. Fíjate en la Figura 1. Se observa la cara del jugador difuminada (*blurred*). ¿Cuáles son las modificaciones a realizar en el fichero *main.html* (en la capa con *id="mystery"*) para que dicha imagen no aparezca difuminada? No se puede añadir nada al fichero, solo se consigue eliminando estilos. La solución de este ejercicio, guárdala en un branch de nombre *unblur* dentro de tu proyecto GitHub.



Tras realizar el ejercicio previo, bájate el fichero zip denominado *milestone1* del proyecto GitHub con nombre *whoareyou*<sup>a</sup>. Rellena las funciones que aparecen ahí (hay tres funciones con el siguiente comentario *YOUR CODE HERE*, *loaders.js#fetchJSON*, *main.js#differenceInDays* y *main.js#getSolution*. Si has realizado todos los pasos correctamente y has programado debidamente, deberías recoger el resultado de la imagen 1.

<sup>a</sup><https://github.com/juananpe/whoareyou/releases/tag/milestone1>

<sup>3</sup>un objeto de tipo Date

<sup>4</sup>Ver: <https://github.com/juananpe/whoareyou/blob/main/js/main.js#L29>

## 7. Adivinando jugadores

Una vez el ordenador ha pensado en el jugador misterioso, el usuario debe adivinar quién es dicho jugador, como mucho en 8 intentos. La aplicación, compara el jugador indicado por el usuario con el jugador misterioso: compara los atributos de los jugadores (edad, liga, puesto, etc.). Con dicho fin, se deberán programar diferentes funciones en los siguientes ejercicios.

### 7.1. Ejercicios

1. Programa la función `getAge(dateString)` en el fichero `js/rows.js`. Como parametro de entrada recoge la fecha de nacimiento de un jugador (como un string ordinario en el formato Año-Mes-Día ¡OJO, en Inglés!) y como resultado devolverá la edad de dicho jugador. Por ejemplo, el resultado de `getAge('1999-01-14')` será 23.
2. Programa la función `check(theKey, theValue)`. Recoge dos parametros , `theKey` y `theValue`. El primero puede ser cualquier atributo (de un jugador) que haya pensado el usuario, el segundo por lo contrario será el valor de dicho atributo. Si coincide con los valores del jugador misterioso, devolvera `correct`, de lo contrario `incorrect`. En el caso de la fecha de nacimiento (`birthdate`), si son de la misma edad `correct`. Sin embargo si la edad es diferente: cuando el jugador misterioso sea mayor, `higher`, y cuando sea menor `lower`.

Por ejemplo, imaginemos que los datos del jugador misterioso son los siguientes:

name: 'Declan Rice', birthdate: '1999-01-14', nationality: 'England', teamId: 1

Entonces:

`check('nationality', 'Spain')` la llamada devolverá `incorrect` (el jugador misterioso no es de España)

`check('birthdate', '1998-01-01')` la llamada devolverá `lower` (dado que el jugador misterioso es menor)

Lo mismo con el resto de atributos (**NOTA:** si se solicita una solución general, es decir, si al jugador le añadimos un atributo nuevo, la función `check` debería seguir funcionando correctamente sin realizar ningún cambio para ello).

Recuerda, que los datos del jugador misterioso están en la estructura de datos `game`, atributo `solution`.

3. Programa la función `getPlayer(playerId)`. Esta función recoge el ID de un jugador y devuelve el objeto JSON que corresponde a dicho jugador.
4. Programa la función `leagueToFlag(leagueId)`. Recoge un número correspondiente a leagueID y devuelve el string correspondiente. En concreto: 564 -> es1, 8 -> en1, 82 -> de1, 384 -> it1, 301 -> fr1. **Importante** (para hacer el ejercicio interesante): NO SE PUEDE usar las cláusulas `if` o `switch`.
5. La función `stringToHTML` está definida de la siguiente manera:

```
const stringToHTML = (str) => {  
  var parser = new DOMParser();  
  var doc = parser.parseFromString(str, 'text/html');  
  return doc.body  
};
```

Introduce dicha función en el fichero `fragments.js` y exportalo. A continuación, importalo desde el fichero `rows.js`.

6. Exporta la función `setupRows` en el fichero `rows.js` (Nótese que se trata de una función anidada, es decir, una función que devuelve otra función). Importa la función `setupRows` en el fichero `main.js`. A continuación, en el fichero `main.js`, en el campo de texto `myInput`, el usuario debe introducir el número de un jugador ( de momento vamos a trabajar con números, más adelante lo modificaremos para trabajar con el nombre) y al pulsar Enter, llamaremos a la función que devuelve `setupRows`. Por ejemplo:

```
let addRow = setupRows( /* parametro bat beharko duzu hemen */ );
...
addRow(jokalariaenIDa);
...
```

Si todo va bien, veremos una animación simple, visulizando los atributos de un jugador (en verde, aquellos que coinciden con los del jugador misterioso, en gris el resto). Ver la Figura 2.

7. Descárgate `milestone2` desde GitHub y rellena los huecos `YOUR CODE HERE` en las mencionadas funciones.

## 8. ComboBox - Lista de opciones

Hasta el momento, el usuario debe introducir el ID del jugador en el campo de texto denominado `myInput`. Dicho campo lo vamos a convertir en una lista de opciones (un combobox). Para ello, eliminaremos de este fichero `main.js` el siguiente fragmento de código:

```
let addRow = setupRows(...);
...
...onkeydown = function (event) {...}
```

En lugar de ello , llamaremos a la siguiente función:

```
autocomplete(document.getElementById("myInput"), game)
```

Dicha función la deberás importar desde el fichero `autocomplete.js`.

A continuación, modificaremos el fichero `autocomplete.js` para poder implementar un *autocomplete combobox*. Nuestro código, se basa en [la respuesta de aquí](#).

¿Qué hace? En resumen, estas son las tareas de la función (siguiendo el código)

1. En el campo de texto `myInput`, genera un *eventListener*, para recoger lo que el usuario teclea ahí (el nombre de un jugador).
2. Prepara un contenedor general, para visualizar el nombre de los jugadores en el objeto con nombre `a`.

3. Recoge los caracteres tecleados por el usuario, y por cada jugador verifica si la función concuerda con el nombre del jugador.
4. En caso afirmativo (en el nombre del jugador aparecen los caracteres tecleados por el usuario), genera un objeto `b` (una capa *div*), con el nombre del jugador (derecha) y con el icono de su equipo (izquierda). Marca en negro los caracteres concordantes.
5. Asigna un Listener al objeto `b`: pulsa sobre ese objeto para llamar al método `addRow`.
6. El combobox se puede usar desde el teclado. Para ello se le asigna un *keydown listener* al objeto `b`.
7. Para finalizar, si el usuario pulsa fuera del combobox, la lista de opciones se debe cerrar (*closeAllLists*).

## 8.1. Ejercicios

Recuerda que en los ficheros `Milestone3`<sup>5</sup> debes buscar el string `YOUR CODE HERE` y a continuación realizar lo que se pide en los ejercicios.

1. Rellena las tres líneas que faltan en la función *Autocomplete* (fichero `autocomplete.js`). En la primera: el código que calcula si el nombre del jugador empieza con el carácter tecleado.
2. En el segundo: dentro del nombre del jugador, hay que marcar en negro la subcadena tecleada por el usuario.
3. En la tercera: llama a la función `addRows` pasándole el ID del jugador elegido por el usuario a través del combobox.
4. No hemos finalizado del todo con el caso de la edad del jugador. El juego, si no adivinamos la edad del jugador misterioso, nos indica si su edad real es mayor o menor a la indicada por el usuario. Pero, ahora mismo no pasa eso. Ver la Figura 2: aparece el número 25 para el caso de Mikel Oyarzabal. Y visualizamos en gris dicho número. Pero, en el juego original, además de eso, nos debe indicar si la edad real del jugador misterioso es mayor o menor a la indicada.

Introduce y exporta las siguientes constantes en el fichero `fragments.js`.

```
const higher = <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 20 20"
  fill="white" aria-hidden="true" width="25" style="margin-right: -8px;
  margin-left: -3px;"><path fill-rule="evenodd" d="M5.293 7.707a1 1 0
0 10-1.41414-4a1 1 0 0 11.414 0 14 4a1 1 0 0 1-1.414 1.414L11 5.414V17a1 1
0 11-2 0V5.414L6.707 7.707a1 1 0 0 1-1.414 0z" clip-rule="evenodd"></
path></svg>;

const lower = <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 20 20"
  fill="white" aria-hidden="true" width="25" style="margin-right: -8px;
  margin-left: -3px;"><path fill-rule="evenodd" d="M14.707 12.293a1 1 0
0 10 1.4141-4 4a1 1 0 0 1-1.414 0 1-4-4a1 1 0 111.414-1.414L9 14.586V3a1
1 0 0 12 0v11.586L12.293-2.293a1 1 0 0 11.414 0z" clip-rule="evenodd"></
path></svg>;

A continuación, modifica la función \textit{setContent} del fichero \hlc[
lightgray]{rows.js}, para que tenga en cuenta la edad, con el fin de
visualizar el icono de lower o higher correspondiente.
```

<sup>5</sup><https://github.com/juananpe/whoareyou/releases/tag/milestone3>

Si todo va bien, como se ve en la Figura 3, deberías tener un combobox al teclear los primeros caracteres del nombre del jugador. Al elegir el jugador, se deben comparar y visualizar los atributos de éste con los del jugador misterioso ( y en el caso de la edad además de viuslizarlos en verde o gris, debe aparecer una flecha hacia arriba o hacia abajo- en caso de no acertar-). Ten en cuenta que, los caracteres acertados se vualizan en negro en todos los nombres. En ese caso existe un problema: en el juego original, los carcateres buscados pueden colocarse en cualquier posición del nombre ( no sólo como prefijo). Esto se deberá arreglar,...

## 9. Intentos y Game Over

El usuario tiene un número máximo de intentos para adivinar: después de haber dado 8 nombres, si el acierto no se ha dado, el juego debe terminar. En un objeto JSON ( al cual lo denominaremos **situacion**), guardaremos un array de intentos (con los ID de los jugadores pronunciados por el usuario) y con la solución (los ID de los jugadores misteriosos).

```
{ "guesses" : [],  
  "solution": IDa }
```

Dicho objeto JSON lo guardaremos haciendo uso del API *localStorage*. En concreto, en una variable de nombre *WAYgameState*.

### 9.1. Ejercicios

Bájate el siguiente modelo de código (<https://github.com/juananpe/whoareyou/releases/tag/milestone4M>) y realiza los ejercicios.

1. Programa la función *initState(what, solutionId)* en el fichero `stats.js`. Recogerá dos valores a modo de parámetro de entrada:

En el parámetro *what*, el ID del objeto que queremos obtener desde *localStorage*.  
*solutionId*: identificador del jugador misterioso.

Como respuesta devolverá un **array** con dos valores:

*state*: si el objeto *what* existe en el objeto *localStrage*, devolverá dicho objeto en la primera posición del array.

*función anonima*: esta función recoge un sólo parámetro denominado *guess*. En este parámetro vendrá el ID del jugador pensado por el usuario. Este mismo será insertado en el array *guesses* concretado en parágrafos anteriores, y se actualizará el objeto *localStorage*.

2. En el fichero `rows.js`, función *setupRows*, llama a la función *initState* y se recogen los valores `[state, updateState]`. En la parte inferior, en la función *addRow* se utiliza la función *updateState()* para guardar en *localStorage* el ID del jugador pensado por el usuario. Todos los ID-s pensados por el usuario están en el array *games.guesses*.

---

<sup>6</sup><https://github.com/juananpe/whoareyou/releases/tag/milestone4>

3. En el ejercicio anterior, no hay que programar nada, sólo entenderlo. Pero una vez entendido, podras programar la función *resetInput* del fichero `rows.js`. Ahí, borra la información que se encuentra en el campo de texto *myInput* y escribe "Guess X of 8", donde X es el número correspondiente al intento actual ( por ejemplo, "Guess 3 of 8").
4. Programa la función *gameEnded*(lastGuess). Ésta recoge el ID del jugador pensado por el usuario y devolverá si el juego ha terminado o no ( el juego finalizará si el ID del jugador misterioso es lastGuess o si se trata del intentro número 8 y no ha adivinado el jugador).
5. En la función *addRow* se han añadido elementos nuevos... Cópialos en tu solución (entre otros la funcion *unblur*)
6. En la función *addRow*, se llama a las funciones *success*() y *gameOver*(), al adivinar la solución o al fallar en el juego respectivamente. Programa estas funciones (de momento son muy parecidas, sólo llaman a la función *unblur*, con os parámetros que les corresponde, para visualizarla imagen de jugador y su nombre.
7. Al hacer Commit y Push, no te olvides de asignarle la etiqueta `Milestone4`.

## 10. Estadísticas

Al terminar el juego, queremos visualizar una ventana de estadísticas (Ver imagen 4). Entre otros, se visualizan 4 números: Total Tries, Success Rate, Current Streak, Best Streak.

- Total Tries. Cuantas veces has jugado.
- Best Streak. La mejor ronda de aciertos. por ejemplo, si aparece 4 significa que has acertado 4 veces seguidas al jugador misterioso (y nunca hs acertado tantas veces seguidas).
- Current Streak. Cuantas veces seguidas (sin fallar) has acertado quién es el jugador misterioso desde que fallaste la última vez.
- Success Rate. Número de aciertos / Total Tries \* 100 (porcentaje)

Dichos datos se almacenarán en *localStorage* en un objeto JSON con la siguiente estructura:

```
{winDistribution: [0,0,0,0,0,0,0,0,0,0],
  gamesFailed: 0,
  currentStreak: 0,
  bestStreak: 0,
  totalGames: 0,
  successRate: 0
}
```



## 10.1. Ejercicios

Teniendo los datos previos, programa las siguientes funciones en el fichero `stats.js`.

Necesitarás bajarte los siguientes modelos de código para realizar estos ejercicios: [Milestone5](#)<sup>7</sup>

1. Recoge el objeto JSON que guarda las estadísticas, y devuleve el valor de `SuccessRate`.
2. La función `getStats(what)`. Recoge como parametro de entrada el nombre del objeto JSON que almacena las estadísticas y que se encuentra en `localStorage`. Si está, devolverá ese mismo objeto, pero si no está, generará el objeto lo inicializará y lo devolverá
3. La función `updateStats(t)`. Se llama a esta función al temrinar el juego ( bien porque el jugador misterioso ha sido adivinado -  $t < 8$  - o bien porque el número de intentos se ha superado -  $t \geq 8$  -). Así, recibe como parametro de entrada el número actual de intentos y en el objeto `gameStats` calculará los siguientes atributos: `totalGames`, `currentStreak`, `gamesFailed`, `windistribution`, `bestStreak`, `successRate`. Para finalizar, se almacenará con el nombre de `gameStats` en `localStorage`.
4. En el fichero `rows.js`, llama a la función `updateStats` (en princio debería de valer con quitar el comentario que aparece)
5. Exporta el fragmento de stats en el fichero `fragments.js`<sup>8</sup> y importalo al fichero `rows.js` (se utilizará la función `showStats`)
6. Con las funciones `gameOver` y `success` llama a la función `showStats`, para visualizar las estadísticas.
7. en el objeto Stats (fichero `fragments.js`) date cuenta que existe una capa con el identificador de `newPlayer`. Nos indica cuando será el nuevo jugador misteriso (en la Figura 4 se visualiza que será dentro de 10 horas 48 minutos y 51 segundos- cuando se cumpla las 0:0:0 el próximo día). Este tiempo se actualiza cada segundo, con ayuda del método `setInterval` ( ver el objeto `interval` en el fichero `rows.js`). Progrma el código que calcula ese tramo de tiempo <sup>9</sup>
8. En GitHub asigne la etiqueta `milestone5` a la version de código escrita hasta el momento en este apartado.

## 11. Mejorando la búsqueda del jugador misterioso

Hasta el momento, al buscar el jugador misterioso, solo marcamos los caracteres que aparecen en las primeras posiciones del nombre (prefijo). Por ejemplo, si escribimos Oyarzabal, en el combobox no aparecerá nada, dado que no hay ningún jugador en el que en el prefijo de nombre tenga el string Oyarzabal. Pero sabemos que existe un jugador de nombre Mikel Oyarzabal. Eso es lo que queremos arreglar en este apartado.

Podemos encontrar un módulo que calcule si dentro de un string hay un substring (y si es así en qué posición) [aquí](#)<sup>10</sup> (con la ayuda de las funciones `match` y `parse`).

[Aquí](#) puedes observar ese módulo en ejecución <sup>11</sup>

---

<sup>7</sup><https://github.com/juananpe/whoareyou/releases/tag/milestone5>

<sup>8</sup>Lo puedes encontrar aquí <https://gist.github.com/juananpe/d224386fc436fcbe1d8449b99f8f7f52>

<sup>9</sup>Aclaración: <https://stackoverflow.com/questions/66277554/how-to-use-intervaltoduration-function-from-date-fns>

<sup>10</sup><https://github.com/moroshko/autosuggest-highlight>

<sup>11</sup><https://codesandbox.io/s/nr994o146m?file=/src/index.js>



El problema es que ese módulo es un módulo de nodeJS y que no se puede ejecutar de manera directa en nuestro navegador. ¿Qué hacer? Por suerte, existe una herramienta denominada browserify. El objetivo de dicha herramienta es realizar las conversiones que permitan ejecutar un módulo de nodeJS en el navegador.

Las explicaciones sobre el comando que realiza dicha conversión las puedes encontrar [aquí](#)<sup>12</sup>.

## 11.1. Ejercicios

### ¡OJO!

Estos ejercicios (milestone6 y milestone7) tienen un nivel de dificultad mayor que los anteriores, y no se ofrecen modelos o plantillas para su realización. Si no encuentras la solución, tómalo con calma, no son obligatorios para aprobar (sólo serán usados para mejorar la nota).

1. Crea los ficheros `match.js` y `parse.js` (deberas ejecutar browserify) para dejarlos preparados para ser usados en el navegador.
2. Modifica el fichero `autocomplete.js`<sup>13</sup> para que use `match` y `parse` (cuando el usuario teclea un substring que aparece en el nombre de un jugador, en el combobox deben aparecer dichos nombres. Por ejemplo, al teclear 'Oyar', debe aparecer en el combobox Mikel Oyarzabal).
3. Si realizas estos ejercicios, marca la versión en GitHub con la etiqueta `milestone6`

## 12. ¿Jugamos otra vez?

Imagínemos que al comenzar el juego y llevar unos cuantos intentos (supongamos 3) debes cerrar el navegador. Al volver, cuando vuelvas a abrir el juego, ¿cómo te gustaría encontrarlo? Seguramente quisieras recuperar el número de intentos realizados previamente y partir de dicha situación ( es decir, desde el intento número 4). O supongamos que has adivinado el jugador misterioso. Cerramos el navegador y a la vuelta quieres visualizar esa victoria. Para darle respuesta a estas dos situaciones, puedes programar la solución que te permita guardar la situación de la partida en *localStorage*. Nada más comenzar el juego, deberíamos checkear en *localStorage* si ya existe algun juego comenzado o no. En caso de existir, la partida debería continuar en las mismas circunstancias en las que se abandonó.

### 12.1. Ejercicios

1. Programa el código que permite guardar la situación del juego. Cierra el navegador y al volver a abrir la página del juego, comprueba en *localStorage* si encuentras algun juego comenzado. En caso afirmativo, recupera la situación. Lo mismo si el juego ya está terminado ( dado que el usuario ha adivinado el jugador misterioso, o dado que ha realizado más de 8 intentos).
2. Si ejecutas este ejercicio, marca en GitHub con la versión `milestone7`

<sup>12</sup><https://stackoverflow.com/questions/28696511/require-is-not-defined-error-with-browserify/41476832#41476832>

<sup>13</sup>Recuerda que aquí tienes una buena aclaración:<https://codesandbox.io/s/nr994o146m?file=/src/index.js>

## 13. Ejercicios opcionales

1. **Favicon.ico.** El navegador busca una imagen de nombre Favicon.ico para colocarlo al lado de cada repisa. Si no lo encuentra, mostrará un error en consola. Para evitar ello, genera en el directorio raíz (/) una imagen `favicon.ico` aunque sea vacío.
2. Para terminar, en el juego original no sólo se visualizan 5 atributos por jugador (país, liga, equipo, posición, edad), si no 6 ya que el número de la camiseta también se tiene en cuenta (igual que con el caso de la edad, explicando si es mayor o menor). Esta información ya está disponible en el fichero JSON de jugadores, por lo que si lo deseas puede tenerla en cuenta y visualizarla en cada fila. (Ver la Figura 6.).
3. Si realizas este segundo ejercicio de libre elección, marca la versión en GitHub bajo la etiqueta `milestone8`.

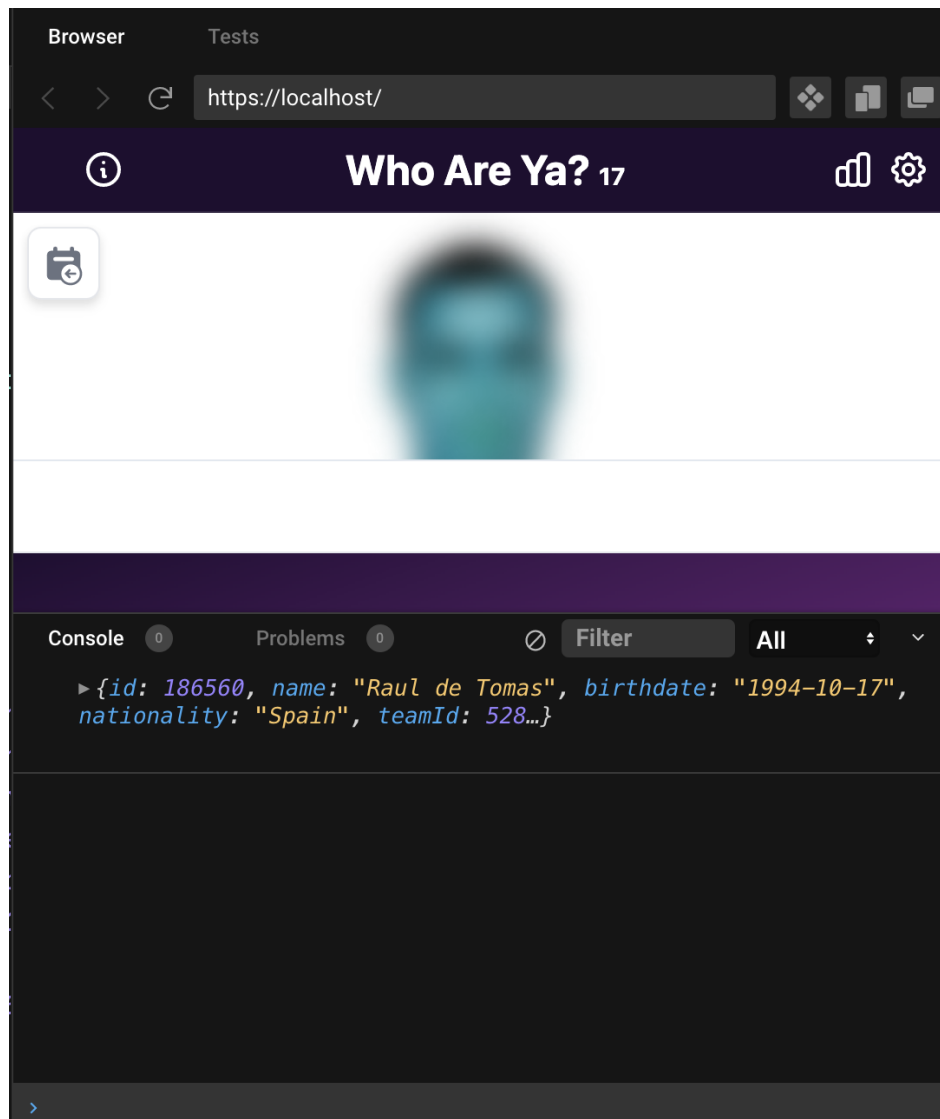


Figura 1: T

ras realizar el ejercicio `Milestone1`, la solución se debería visualizar de esta manera en el navegador. Date cuenta, que en consola aparecen los datos relacionados con el jugador misterioso, dicha información te será útil a la hora de programar. Pero al terminar no te olvides de eliminar dicho log.

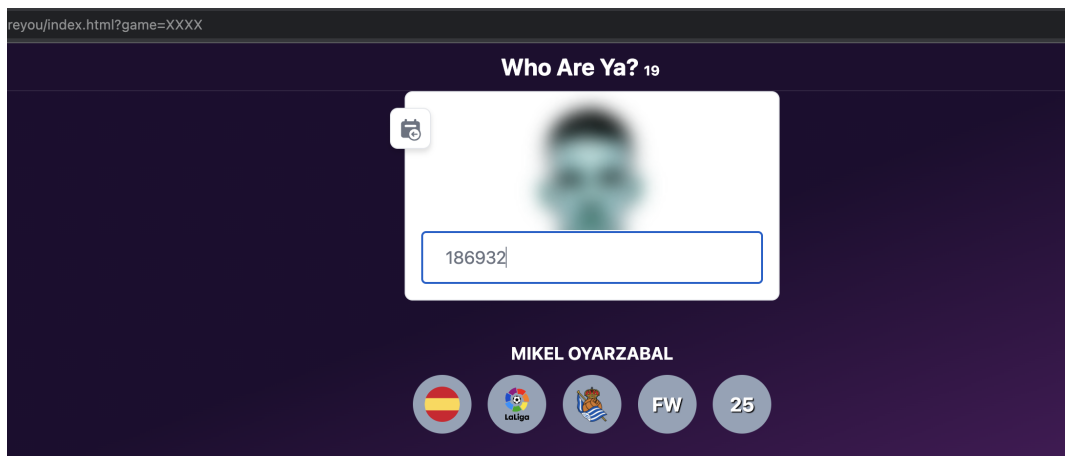


Figura 2: tras realizar los ejercicios de Milestone2, la solución se deería de visualizar de esta manera en el navegador.

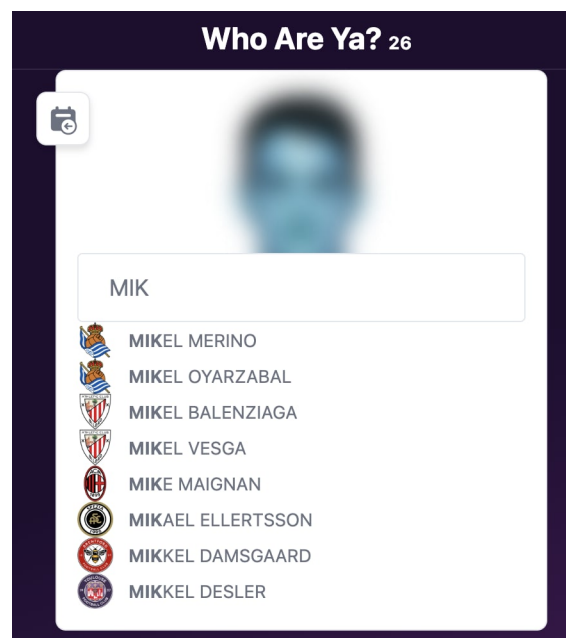


Figura 3: Tras completar los ejercicios Milestone3, la solución se debe visualizar de esta manera en el navegador

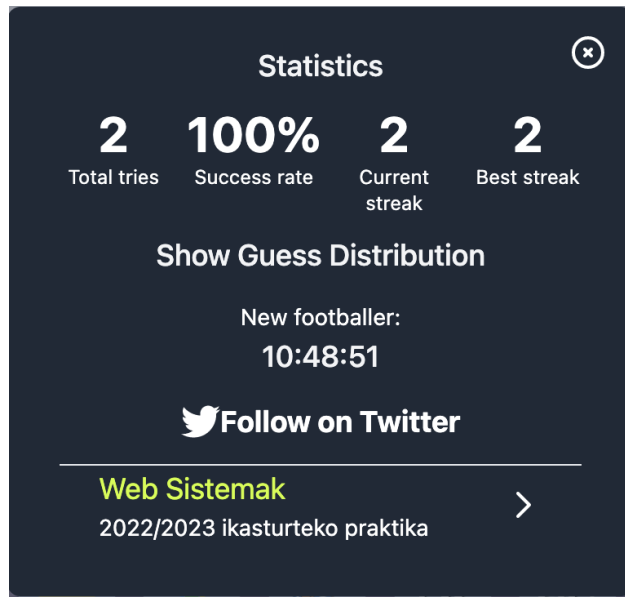


Figura 4: Al temrinar ek juego se deben visualizarlas estadísticas

✖ Failed to load resource: the server responded with a status of 404 (Not Found)

Figura 5: El navegador buscará una imagen de nombre Favicon.ico para colocar al lado de cada repisa

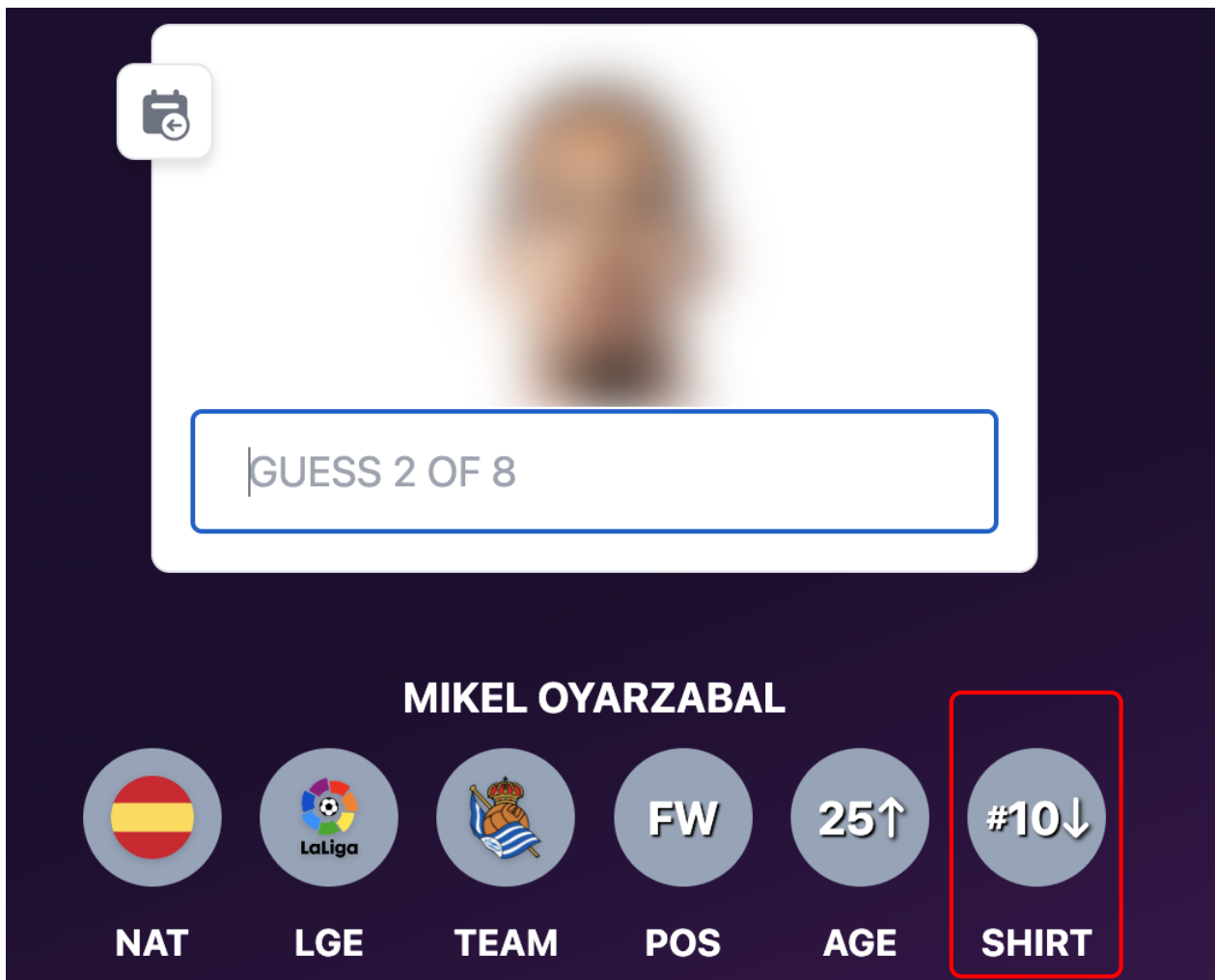


Figura 6: El número de la camiseta (*shirt number*) del jugador también estaría bien