

UNIVERSIDAD AUTÓNOMA DE MADRID

DEPARTAMENTO DE INFORMÁTICA

---

# Proyecto de Sistemas Informáticos

## Práctica - 4

---

Roberto MARABINI  
Alejandro BELLOGÍN

## Registro de Cambios

Versión <sup>1</sup>	Fecha	Autor	Descripción
1.0	10.08.2022	RM	Primera versión.
2.0	26.10.2022	RM	Cambio heroku → render.com
2.1	5.12.2022	RM	Renumerar práctica 5 → 4
2.2	22.12.2022	AB	Traducción al inglés
2.3	29.1.2023	RM	Revisión antes de subir la guía a <i>Moodle</i>

---

<sup>1</sup>La asignación de versiones se realizan mediante 2 números  $X.Y$ . Cambios en  $Y$  indican aclaraciones, descripciones más detalladas de algún punto o traducciones. Cambios en  $X$  indican modificaciones más profundas que o bien varían el material suministrado o el contenido de la práctica.

## Índice

<b>1. Objetivos</b>	<b>3</b>
<b>2. Trabajo a realizar durante la primera mitad de la práctica</b>	<b>3</b>
2.1. Implementación de un API Rest . . . . .	3
<b>3. Trabajo a desarrollar durante el resto de la práctica</b>	<b>5</b>
3.1. Implementación . . . . .	6
3.2. git . . . . .	7
<b>4. Trabajo a presentar al finalizar la práctica</b>	<b>8</b>
<b>5. Criterios de evaluación</b>	<b>9</b>
<b>A. Imágenes</b>	<b>13</b>

## 1. Objetivos

En esta práctica se finalizará de implementar la aplicación *kahootclone* desarrollando los métodos necesarios para que los participantes puedan responder a las preguntas de los cuestionarios. Os recordamos que en esta práctica SI valoraremos la estética de la aplicación web así como su usabilidad. Queda a vuestra elección el aspecto y diseño de la aplicación sólo os solicitamos que visualmente NO se parezca a la que os damos como referencia.

## 2. Trabajo a realizar durante la primera mitad de la práctica

El trabajo a realizar parte del código desarrollado en la práctica anterior así que se recomienda que sigáis usando el mismo repositorio privado que usasteis en la última práctica PERO cread otra rama (**branch**) llamada *práctica\_4* para aislar el código de la práctica 3 y de la práctica 4. Por otra parte, cuando despleguéis la aplicación en *render.com*, crea un proyecto y una base de datos nuevos y no pises el usado en la práctica anterior ya que será necesario para corregir la práctica anterior.

En la práctica anterior se utilizó el entorno de *Django* para crear el GUI de los usuarios de la aplicación, en esta práctica usaremos *Vue.js* para crear el interfaz gráfico al que accederán los participantes en lugar de *Django*. *Vue.js* tendrá acceso a la base de datos donde se guardan los modelos mediante un API Rest que construiremos usando el *Django Rest Framework*. Las llamadas se realizarán usando el módulo *fetch* o el módulo *Axios* de *Vue.js*.

### 2.1. Implementación de un API Rest

Puedes reutilizar el proyecto de *django* creado en la práctica anterior y añadirle una nueva aplicación llamado **restServer** donde se alojará todo el código que se describe a continuación. Nos harán falta tres métodos (podéis ver parcialmente el resultado de implementar los mismos en <https://kahooclone.onrender.com/api/>). El primero servirá para unirse al juego. El segundo implementará la espera hasta

que todos los participantes se hayan unido al juego y el tercero servirá para enviar respuestas (`guess`) a las preguntas propuestas. A continuación se describen los servicios en detalle. Recordad que estos son servicios Rest y que por lo tanto reciben y devuelven un diccionario en formato JSON. Para cada servicio se proporciona el URL donde deben escuchar, un/unos test disponibles en el repositorio (véase fichero `djangoRestServer/tests.py`) (se proporcionan las 6 primeras letras de cada conjunto de test) y una descripción que incluye el “input” y “output” esperado.

`/api/participant/` — `test02...` — Recibe un diccionario `{'game': int, 'alias': cadena}` conteniendo los valores de `game.publicID` y `participant.alias`. El método crea un participante (con el alias y el `game` recibidos) y devuelve el participante creado el cual contiene el identificador `uuidP` que será usado en el futuro para identificar al participante. Nótese que en las APIs Rest no se suele poder usar las variables de sesión.

`/api/games/` — `test01...` — Recibe un diccionario `{'publicId': int}` conteniendo un `game.publicId` y devuelve un objeto `game` serializado. Sirve para comprobar cuando cambia el estado del juego (`game.state`) de `WAITING` a `QUESTION` lo que ocurre cuando acaba el tiempo destinado a que los participantes se unan al juego y se pulsa el botón que inicia el juego.

`/api/guess/` — `test03...` — Recibe un diccionario `{'game': int, 'uuidp': string, 'answer': int}` con un `game.publicId`, un `participant.uuidP` así como un número entero en el rango `[0 – 3]` que almacena la respuesta seleccionada por el participante. Con esta información se crea un `guess`. El método admite la creación de un único `guess` por cada tupla (participante, pregunta), igualmente comprueba antes de añadir el `guess` que el participante existe y el estado del juego (`game.state`) es `QUESTION`.

En todos los casos se deben gestionar los errores más evidentes como puedan ser:

- La elección del mismo alias por dos participantes. En este caso se devolverá un mensaje de error.

- Impedir que el mismo participante mande dos respuestas a una pregunta. Se admitirá sólo la primera respuesta y se mostrará un mensaje de error en el segundo (o sucesivos) intento.
- Envío de identificadores incorrectos. Por ejemplo identificadores de juego que ya se han cerrado, de participantes que no están asociados a juegos abiertos o inexistentes.

Nota: Por defecto la API Rest de *Django* crea un conjunto de métodos sin restricciones de acceso. Es tu responsabilidad cuidar de que no se pueda acceder a la base de datos de modelos para realizar funciones diferentes a las descritas en este apartado. Existen varias maneras de conseguir este objetivo, en la implementación que mostramos en <https://kahoorclone.onrender.com> se consiguió un control más fino de los permisos de acceso redefiniendo el método `get_permissions`. Un efecto negativo de restringir los permisos es que se reduce drásticamente la utilidad del interfaz de prueba <https://kahoorclone.onrender.com/api/> por lo que puede ser interesante crear una primera versión del API sin restringir los permisos hasta que el interfaz funcione adecuadamente.

**Testing** Para verificar la correcta implementación del API Rest se ha definido en el fichero `djangoRestServer/tests.py` una batería de test (no necesariamente completa) que debe satisfacer tu código. Estos test deben entenderse como requerimientos adicionales al proyecto.

### 3. Trabajo a desarrollar durante el resto de la práctica

Durante esta segunda mitad de la práctica se implementarán las páginas web que consumirán los servicios Rest y se desplegará la aplicación en *render.com* tal y como se describió en la práctica anterior. Recordad que el despliegue debe hacerse en modo de producción. Igualmente se debe escribir un manual de usuario. Un manual de usuario debe proporcionar instrucciones de cómo utilizar vuestra aplicación web pero

no debe describir como la habéis implementado, que problemas habéis encontrado durante la implementación o cualquier otro detalle relacionado con la implementación. Concentraros en el uso de la aplicación tanto en la creación de cuestionarios como en el juego con ellos.

Los tres métodos descritos en el apartado anterior deben ser llamados por tres páginas creadas en *Vue.js* usando *fetch* (véase <https://kahootclone-render-vue.onrender.com>). En este párrafo se usa el termino página de forma liberal puesto que por tres páginas nos referimos a tres URLs definidos por el *router* de *Vue.js*. La primera página mostrará un formulario preguntando el `participant.alias` y el `game.publicId` y llamará a `/api/participant` para crear el nuevo participante (Fig. 1). La segunda mostrará un mensaje al participante comunicándole que se está a la espera de que el juego empiece (Fig 2). Esta página debe comprobar el estado del juego `game.status` cada 2 segundos llamando a `/api/games` y cuando `game.status` pase de `WAITING` a `QUESTION` se pasará a la tercera página (Fig. 3). Esta última página permitirá elegir entre las cuatro respuestas posibles (un número en el rango  $[0 - 3]$ ) y enviará la elección al servidor llamando a `/api/guess`. Tras mandar una respuesta se debe recibir un mensaje confirmando el envío o un error en caso de que el envío no haya sido satisfactorio.

### 3.1. Implementación

Cread un proyecto llamado *vueClient* con el comando `npm init vue@3.2` (alternativamente se puede usar `make init` usando el *makefile* existente en el directorio *vueClient*). Este comando inicializa la versión 3.2 de *vue* usando la utilidad *vite*. Elegid las opciones siguientes (marcadas con dos asteriscos, esto es, *\*XX\**):

```
# Project name: ... vueClient
# Add TypeScript? ... *No* / Yes
# Add JSX Support? ... *No* / Yes
# Add Vue Router for Single Page Application development? ... No / *Yes*
# Add Pinia for state management? ... *No* / Yes
# Add Vitest for Unit Testing? ... *No* / Yes
# Add Cypress for both Unit and End-to-End testing? ... *No* / Yes
```

```
# Add ESLint for code quality? ... No / *Yes*  
# Add Prettier for code formatting? ... No / *Yes*
```

y ejecutad

```
cd vueClient  
npm install  
make requirements
```

Estos comandos instalarán los siguientes módulos

```
bootstrap@5.1.3  
@popperjs/core@2.11.5  
axios@0.27.2  
vuex@4.0.2
```

Recordar que `npm run dev` (o `make runserver`) levanta el servidor y `npm run lint` (o `make lint`) ejecuta el programa `lint`. `axios` es una librería similar a `fetch`, `vuex` sirve para crear variables de sesión y `bootstrap` da acceso al framework de CSS `bootstrap`. No tenéis obligación pero podéis usar estas librerías si os resultara conveniente.

A continuación implementad las tres páginas solicitadas siendo la “homepage” aquella que muestra un formulario preguntando por el `participant.alias` y el `game.publicId`.

### 3.2. git

Como de costumbre guardad y compartid vuestro código usando *git*. `make init` y `make install` van a instalar una colección de módulos en el directorio `node_modules`. No subáis este directorio al repositorio puesto que el contenido depende de vuestro sistema operativo.



## 4. Trabajo a presentar al finalizar la práctica

- Aseguraros que vuestro código implementa toda la funcionalidad solicitada y que satisface todos los test proporcionados. No es admisible que se modifique el código de los test excepto las variables situadas al principio del fichero las cuales se encuentran claramente identificadas.
- Implementad todos los test que consideres necesarios para cubrir (coverage) la funcionalidad desarrollada. Estos test se deben implementar en un fichero llamado `tests_additional.py`.
- incluid en la raíz del proyecto un fichero llamado `coverage.txt` que contenga el resultado de ejecutar el comando `coverage` para todos los test.
- incluid en la raíz del proyecto un fichero llamado `user_guide.pdf` que contenga el manual de usuario.
- Desplegad y probad la aplicación en *render.com* en modo de producción.
- Esta practica requiere presentar 2 proyectos: uno en *Django* que gestiona la parte administrativa y además contiene el API REST y uno en *Vue.js* al que se conectan los participantes. Subid a *Moodle* dos ficheros obtenido al ejecutar el comando `zip -r ../xxxxx.zip .git` desde la raíz de los proyectos de *Django* y *Vue.js* respectivamente. Cambiar `xxxxx` por *Django* o *Vue.js* según el proyecto que contengan. Recordad que hay que añadir y “comitir” los ficheros a git antes de ejecutar el comando. Si queréis comprobar que el contenido del fichero zip es correcto lo podéis hacer ejecutando la orden: `cd ..; unzip assign5_final.zip; git clone . tmpDir; ls tmpDir`.
- Aseguraros de que la variable `ALLOWED_HOSTS` del fichero `settings.py` incluido en la entrega contiene la dirección de despliegue en *render.com*.

Igualmente se espera que las variables **Cross-Origin Resource Sharing** (**CORS\_ORIGIN\_WHITELIST**) estén adecuadamente inicializadas para poder ejecutar la aplicación en *render.com*. Finalmente, comprobar en *render.com* que tanto el nombre de usuario como la clave del usuario de administración son *alumnodb*. Si cualquiera de los dos URL necesarios para ejecutar la práctica en *render.com* no están disponibles tal y como se indica en este punto o la password del usuario de administración no es la solicitada se calificará la práctica como si no se hubiera desplegado en *render.com*.

- Comprueba que a la hora de escribir ficheros en Python, se es coherente con los criterios de estilo marcados por *Flake8*. *Flake8* no debe devolver ningún error al ejecutarse sobre las líneas de código programadas por el estudiante.
- Igualmente los ficheros creados con *Vue.js* deben satisfacer los criterios marcados por *lint* (comando `npm lint`)

## 5. Criterios de evaluación

Para aprobar con 5 puntos es necesario satisfacer en su totalidad los siguientes criterios:

- Es posible crear un cuestionario y jugar con él localmente. Se muestra la puntuación de los participantes tras cada pregunta.
- El fichero subido a *Moodle* contiene un repositorio de git.
- El código se ha guardado en un repository git accesible a todos los miembros de la pareja y este repositorio es privado.
- Al ejecutar los test el número de fallos no es superior a diez y el código que los satisface es funcional.

Cumpliendo los siguientes criterios se puede optar a una nota máxima de 5.9:

- Los criterios enunciados en el párrafo anterior se satisfacen en su totalidad.
- Al ejecutar en local los test el número de fallos no es superior a seis y el código que los satisface es funcional.
- No es posible crear/borrar/editar cuestionarios/preguntas/respuestas para usuarios NO conectados (“logueados”) aunque se acceda directamente a los URLs asociados con los diferentes servicios.
- No es posible unirse a juegos, o crear resultados (**guess**) para participantes que desconozcan la **publicId** del juego aunque se acceda directamente a los URLs asociados con los diferentes servicios.

Cumpliendo los siguientes criterios se puede optar a una nota máxima de 6.9:

- Los criterios enunciados en el párrafo anterior se satisfacen en su totalidad.
- Los dos proyectos solicitados están desplegados en *render.com*. En el fichero **settings.py** se encuentra añadida la dirección del primer proyecto en *render.com* a la variable **ALLOWED\_HOSTS** y la del segundo a **CORS\_ORIGIN\_WHITELIST**. Además de estar desplegados, los dos proyectos funcionan correctamente en *render.com*.
- La aplicación de administración de base de datos (admin/) se ha desplegado y está accesible en *render.com* usando el username/password *alumnodb*.
- La aplicación se ha desplegado en *render.com* en modo de producción.
- Usando la aplicación de administración es posible crear, listar o borrar objetos pertenecientes a todos los modelos solicitados.

Cumpliendo los siguientes criterios se puede optar a una nota máxima de 7.9:

- Los criterios enunciados en el párrafo anterior se satisfacen en su totalidad.
- En *Django* se ha usado herencia a la hora de crear las paginas web.

- Todo formulario que o bien esté relacionado con un modelo o se solicite expresamente en el enunciado, ha sido construido usando el sistema de formularios implementado por *Django*.
- Se han usado ficheros css o entornos de tipo *bootstrap* para definir los estilos.
- Al ejecutar los test el número de fallos no es superior a cuatro y el código que los satisface es funcional.
- La aplicación web tiene un interfaz de usuario visualmente cuidado y agradable al uso. Visualmente la aplicación web NO debe parecerse al modelo que os hemos dado en *render.com*.

Cumpliendo los siguientes criterios se puede optar a una nota máxima de 8.9:

- Los criterios enunciados en el párrafo anterior se satisfacen en su totalidad.
- El código es legible, eficiente, está bien estructurado y comentado.
- Se utilizan las herramientas que proporciona el framework.
- Sirva como ejemplo de los puntos anteriores:
  - Las búsquedas las realiza el sistema gestor de la base de datos. Esto es, no se traen todos los elementos de una tabla (i.e. `ClassName.objects.all()`) y se busca en las funciones definidas en `views.py` (`for object in class: if object.name=='Pedro': ...`).
  - Los errores se procesan adecuadamente y se devuelven mensajes de error comprensibles.
  - La aplicación web gestiona adecuadamente accesos a páginas inexistentes o con permisos inadecuados.
  - El código presenta un estilo consistente y las funciones están comentadas incluyendo su autor. Nota: el autor de una función debe ser único.
  - A la hora de escribir ficheros en python, se es coherente con los criterios de estilos marcados por *Flake8*. *Flake8* no devuelve ningún error al ejecutarse sobre las líneas de código programadas por el estudiante.

- Los ficheros creados con *Vue.js* satisfacen los criterios marcados por *lint* (comando `npm lint`).
- Se ha incluido el manual de usuario y es adecuado.
- Al ejecutar los test el número de fallos no es superior a dos y el código que los satisface es funcional.
- Si reducimos el tamaño de la ventana del navegador o utilizamos el zoom, todos los elementos de la página siguen resultando accesibles y no se pierde funcionalidad.

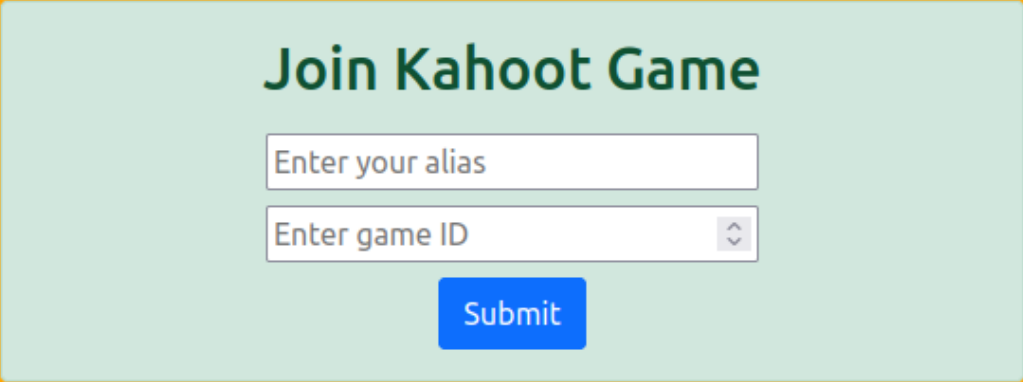
Cumpliendo los siguientes criterios se puede optar a una nota máxima de 10:

- Los criterios enunciados en el párrafo anterior se satisfacen en su totalidad.
- La aplicación es robusta y responde adecuadamente incluso si se proporcionan parámetros inválidos.
- Todos los test dan resultados satisfactorios.
- La cobertura para los ficheros de *Django* que contienen los modelos, las vistas y los formularios es superior al 99 %. Si los test propuestos no generan la cobertura necesaria cread un fichero nuevo llamado `tests_additional.py` y añadid los test necesarios en el mismo.

Nota: Entrega final fuera de plazo → substraer un punto por cada día (o fracción) de retraso en la entrega.

Nota: El código usado en la corrección de la práctica será el entregado en *Moodle*. Bajo ningún concepto se usará el código existente en *render.com*, *Github* o cualquier otro repositorio.

## A. Imágenes

The image shows a web form for joining a Kahoot game. It has a solid orange background. In the center is a light green rounded rectangle containing the text "Join Kahoot Game" in a bold, dark green font. Below this text are two white input fields. The first field is labeled "Enter your alias" and the second is labeled "Enter game ID". The second field has a small dropdown arrow on its right side. Below the input fields is a blue button with the word "Submit" in white text.

**Join Kahoot Game**

Enter your alias

Enter game ID

Submit

Figura 1: Página usada por los participantes para unirse a un juego.

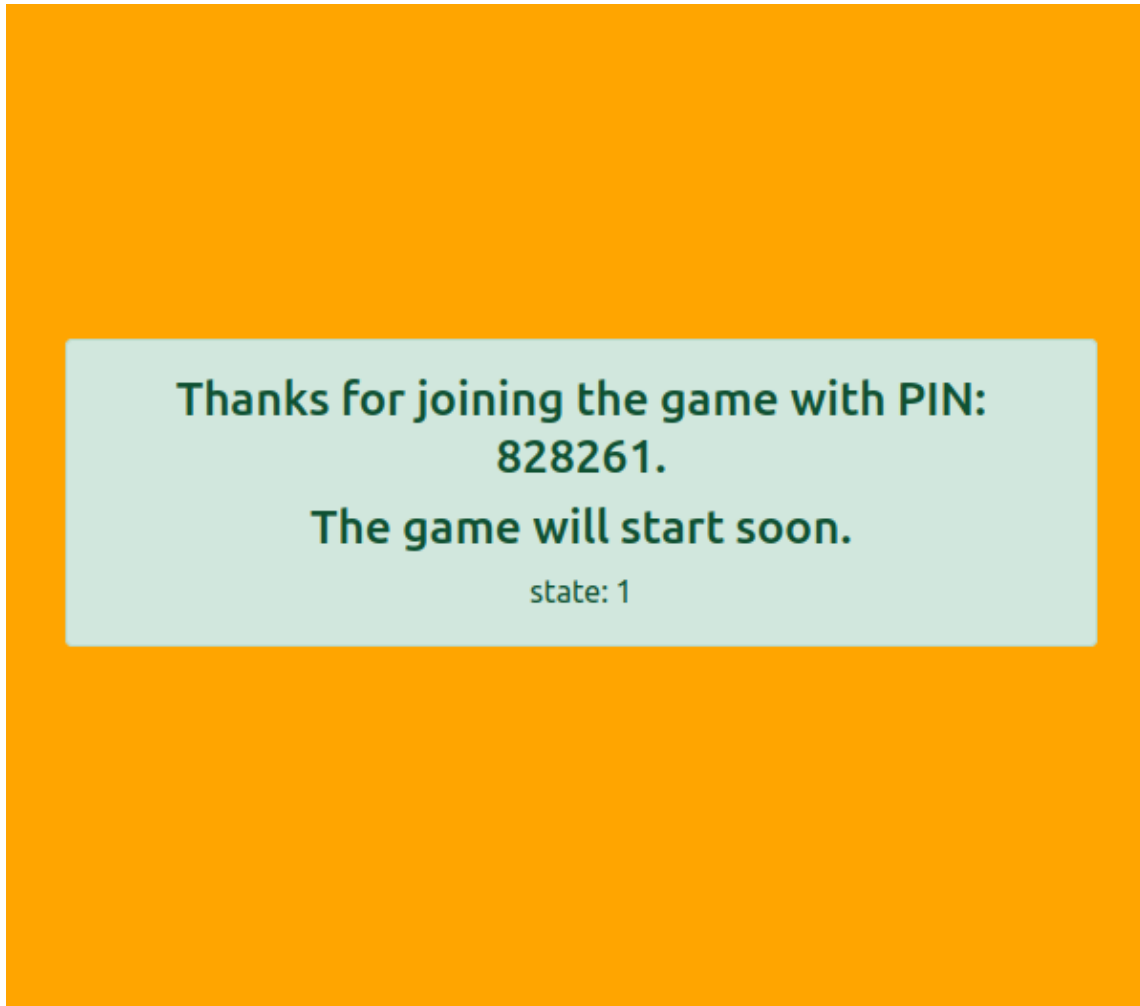


Figura 2: Página de espera hasta que el juego comience.

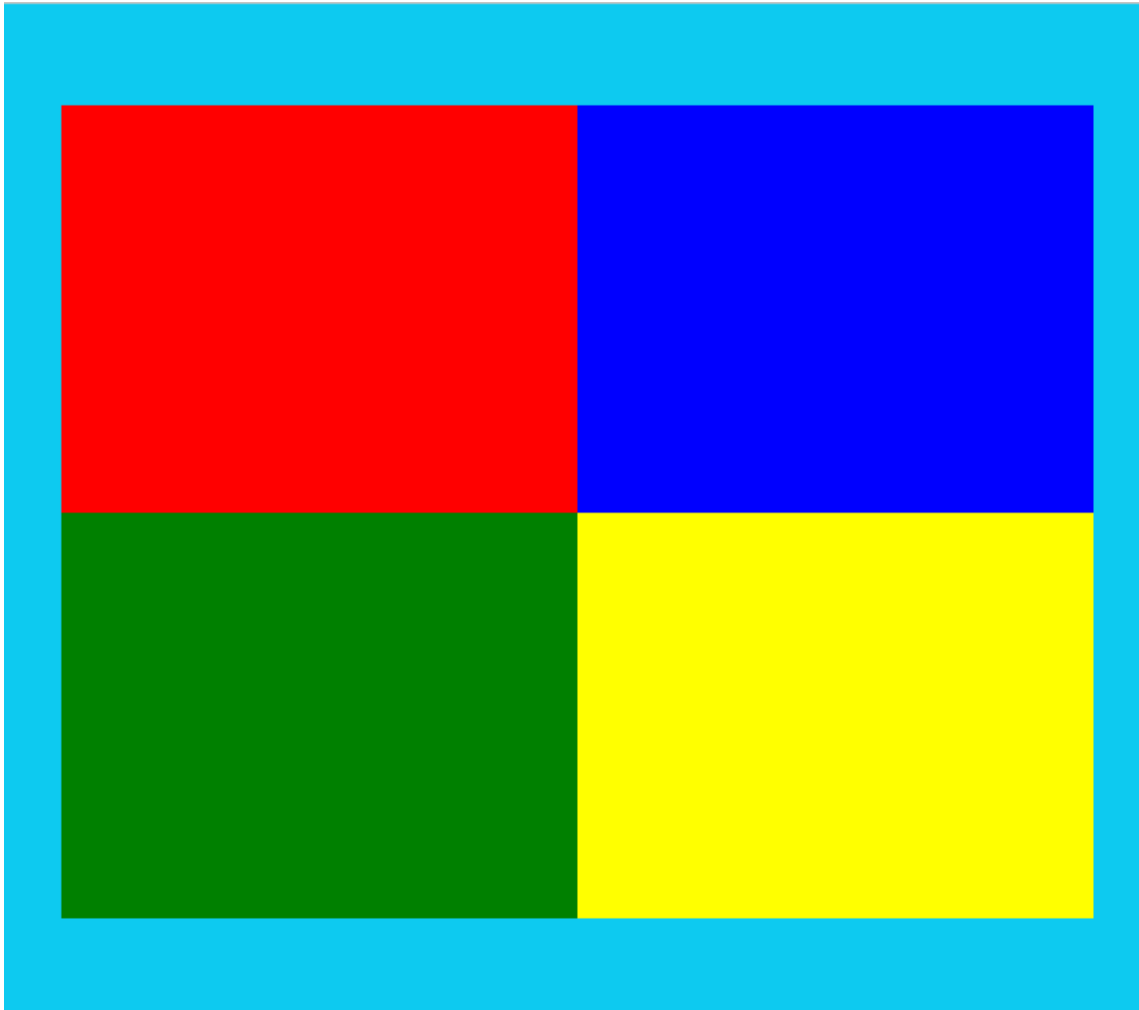


Figura 3: Página usada para seleccionar una respuesta.