# Documentación Servidor de Navidad

## Arquitectura de salas comunicación

El servidor utiliza la librería channels de Django para poder realizar de forma eficiente los avisos entre clientes mediante sockets. Esto se aplica cuando uno o más servicios dependen de que otro u otros hayan terminado. Esto se resuelve creando una sala donde se espera a recibir la notificación de que se puede continuar. Hay 2 tipos:

* Wait Room: se usa en todos los juegos para que los jugadores vayan automáticamente al juego que esté listo para unirse. Los jugadores conectados están esperando a que haya un juego que jugar. El admin se une a ella para avisar de que hay un juego listo.
* In Game Room: se usa en aquellos juegos donde es obligatoria la participación de todos los jugadores para avisar al servidor de que todos los jugadores ya han participado en el juego y que se puede proceder con el siguiente paso. Esto ocurre en el juego de la ruleta, en el ahorcado y en el pistolero rey de la pista. El admin se une a ella cuando el juego está listo y los jugadores se unen fugazmente a ella cuando han mandado su participación en el juego. Cada vez que ocurre, el servidor comprueba si era el último jugador que faltaba y en ese caso, avisa al admin de que continúe. Esta lógica es distinta en el pistolero, ya que solo es necesario que dos jugadores manden la notificación.

Estas salas se configuran creando clases de consumers (que es la manera de implementarlo en Django), en consumers.py, en este caso, una para cada sala. Además, en el archivo routing.py hay que crear una ruta para cada una (para que se llame desde JS).

## Variables de control

Estas sirven para ser conscientes de cómo manejar a los jugadores.

* *can\_players\_join*: se guarda en el servidor y se consulta desde JS para saber si hay un juego en curso al que unirse. Si es true, abre la posibilidad de unirse al juego desde wait\_room.html.

Se cambia con **set\_can\_players\_join(bool)** desde JS o Python.

* + Cambiarla a true: lo hace el servidor cuando empieza el siguiente juego. Esto puede pasar desde dos lugares:
    - Depende de la decisión de la función **transition\_to\_next\_game(),** que solo la activa si hay un juego programado.
    - Desde **set\_game(id, rounds)** que se llama desde **set\_next\_game(id)** en JS por el admin cuando selecciona juego en game\_selector.html.
  + Cambiarla a false:
    - Se crea como false cuando se instancia el servidor.
    - El admin desde el JS de cada juego cuando termina (a la vez que *can\_players\_interact* en los juegos correspondientes*)*
* *can\_players\_interact:* se guarda en el servidor y la usa el servidor para ver si hace caso a una acción ejecutada por un jugador en alguno de los juegos (se añade como un condicional al principio de esas funciones).

Se cambia con **set\_can\_players\_interact(bool)** desde JS o Python.

* + Cambiarla a true: lo hace el admin cuando quiere activar que los jugadores puedan interactuar (desde **listen\_in\_game\_room** o directamente desde el bucle del juego si no hay Game Room).
  + Cambiarla a false:
    - Se crea como false cuando se instancia el servidor.
    - El admin desde el JS de cada juego cuando quiere que no haya más interacción (de momento a la vez que *can\_players\_join)*
* *played*: se guarda en la sesión de JS de cada jugador.
  + Cambiarla a true: cada uno de los clientes cuando acaba de jugar y se le va a mandar a wait\_room.html.
  + Cambiarla a false:
    - Se crea como false cuando se registra el cliente en login.html con **register\_name().**
    - Se cambia a false justo antes de entrar a un juego por la función **play\_game(id).** Esta se llama desde **play\_next\_game()**, que se llama desde la wait\_room.html

## Redirecciones en la Wait Room

Las redirecciones se gestionan con funciones de comprobación contenidas en game\_redirects.js. La mayoría se conectan con funciones homónimas en Python (main\_views.py que llama a la función homónima del main\_controller.py).

Funciones principales

**play\_next\_game()** se llama desde la pantalla wait\_room.html al llegar. Está diseñada para manejar los dos motivos por los que el jugador puede encontrarse en ella según el valor de la variable de sesión *played*:

* *Played*=true: ha sido enviado por un juego cuando ya ha jugado a él.
* *Played*=false: el jugador ha acabado aquí por un flujo fuera del normal (se le ha desconectado el movil, etc). Se le redirige al lugar correcto en función del valor de la función id = **get\_ready\_to\_join\_game().**
  + Si devuelve un código: llama a **play\_game(id),** que pone *played*=false y lleva a la pantalla del juego actual.
  + Si devuelve -1: quedarse en la wait room de sockets con la función **listen\_in\_wait\_room()**

**admin\_next\_game():** llamada por el admin al terminar la pantalla de ranking\_and\_prizes.js. Llama a la función del servidor **transition\_to\_next\_game()** que devuelve el siguiente game id según la programación (y activa la variable *can\_players\_join* en el servidor en ese caso):

* Si existe: se carga ese juego desde admin con **admin\_game(id),** que llama a **notify\_clients\_game\_ready(id)** para que los clientes vayan a la pantalla correspondiente.
* No existe: se carga game\_selector.html

**set\_next\_game(id)** llamada por el admin cuando selecciona juego en game\_selector.html. Configura el juego con las rondas correspondientes elegidas con la view de Python **set\_game(id, rounds)** y llama a **admin\_game(id)** quecarga la pantalla del juego del admin y llama a **notify\_clients\_game\_ready(id).**

Funciones auxiliares (se llaman desde las funciones principales)

* **set\_can\_players\_join(bool)** cambia el valor de *can\_players\_join.*
* **set\_can\_players\_interact(bool)** cambia el valor de *can\_players\_join.*
* **get\_ready\_to\_join\_game()** si *can\_players\_join = true* esta devuelve el código del juego y si no, -1.
* **admin\_game(id)** la llama el admin para ir a la pantalla del juego correspondiente notificar a todos los clientes (con para que vayan a la pantalla de juego correspondiente.
* **play\_game(id)** lleva a un jugador a la pantalla correspondiente al juego y pone *played*=false. Es llamada en los dos casos en los que sabe que hay un juego:
  + Llegas a wait\_room.html y ya hay un juego activo. La función **play\_next\_game()** lo detecta y te redirige.
  + Estás escuchando en la wait room (con **listen\_in\_wait\_room())** y te notifica el admin (con **notify\_clients\_game\_ready(id)).**

### Cómo se gestiona la conexión a una sala

Para ello, se llama desde JS al servidor para unirse, salir, escuchar o mandar mensajes a la sala.

**notify\_clients\_game\_ready(id)** llamada por el admin cuando el servidor decide que hay un nuevo juego (o automático desde **admin\_next\_game()** o elegido desde **set\_next\_game(id)**). Une al admin a la sala con **join\_wait\_room(),** y manda el id del juego a todos llamando a la función **receive()** de consumers.py y se sale cerrando el socket. Esto tiene más pasos, ya que la función llama a **send\_game(),** la cual llama **send\_game\_type().**

**listen\_in\_wait\_room()** llamada por el cliente cuando accede a wait\_room.html y no hay nada que jugar. Une al cliente a la sala con **join\_wait\_room(),** la cual le da un socket abierto y después se queda esperando para a que el admin le llame con **send\_game\_type()** que le dará un id para unirse a un juego. Cuando pasa, se sale de la sala cerrando el socket, llama a **play\_game(id)** que le lleva a la pantalla correspondiente y pone *played*=false.

**join\_wait\_room()** esta función es llamada por las dos de arriba para unirse a la wait room, pero no se llama nunca directamente. Esta se gestiona en consumers.py por la función **connect(),** que llama a **join\_room()** para configurar el socket de conexión y añadirlo a la sala para que **connect()** lo acepte y lo devuelvaa JS.

Cuando el usuario se desconecta cerrando el socket desde las funciones anteriores se llama a **disconnect()** en consumers.py**.**

## In Game Room

Se gestiona desde las funciones de JS que llaman el admin o los jugadores para notificar al servidor con sus funciones homónimas. Funciona de una manera muy similar a la wait room, solo diré lo que es distinto.

**listen\_in\_game\_room()** llamada por el admin justo cuando quiere que los jugadores puedan iteraccionar. Pone *can\_players\_interact=* true, se une a la sala con **join\_wait\_room()** y se queda esperando a que le llegue el mensaje de **send\_last\_player\_type()** de consumers.py. Cuando pasa, cierra el socket para salir de la sala pone *can\_players\_interact=* false y prosigue con el juego (termina o siguiente ronda).

**notify\_player\_has\_interacted()** llamada por cada jugador cuando juega al juego (resultado del botón de enviar) para avisar al admin. El jugador se une a la sala con **join\_in\_game\_room()**, lo cual es gestionado por la función de consumers.py **receive()**, quellama a la función **get\_remaining\_interactions()** del main\_controller.py. Si esta le dice que el número de jugadores que quedan por interactuar es 0, llama a la función de consumers.py **send\_last\_player()** que llama a **send\_last\_player\_type()** que es la que notifica al admin (realmente a toda la sala, pero los jugadores no han llamado a **listen\_in\_game\_room()).**

**join\_in\_game\_room()** es llamada por admin o jugadores para unirse a la sala. Funciona igual que en la wait\_room.

## Uso de In Game Room

Todos usan la Wait Room, pero la In Game Room solo es necesaria si queremos garantizar que todos los jugadores hayan participado en el juego antes de proceder a evaluar los resultados para evitar que alguno se quede sin jugar. Esto se hace en alguno de los juegos donde se considera que metería prisas a los jugadores y podría estropear la experiencia.

* Ruleta: tiene sentido, ya que es un juego rápido donde no hace falta meter prisa.
* Democracia: va en contra de la definición del juego: el servidor recoge los mensajes a petición del admin una vez por segundo, lo que haya entrado en ese segundo es lo que ha entrado.
* Panel: tras es rework ya no va por pasos, así que no es necesario.
* Pistolero: se ha decidido usar la sala para asegurarse de que el mensaje no se pierde en el timing. el juego da la sensación de que puede ser así e incentiva a ambos jugadores a darle al final de la cuenta atrás, así que no es problema.
* Blind Number: va en contra de la definición, ya que la participación es voluntaria.

Posible creación de una nueva sala:

En la ruleta y el el rework del ahorcado (panel) y en el blind number sería interesante que cuando un jugador interacciona el admin reciba un mensaje de que actualice la pantalla inmediatamente con la nueva información. Puede ser más fácil que el admin pregunte una vez cada medio segundo y ya, ya que es solo una cuenta y no añade mucho tráfico. Si se implementa la sala habría que tener cuidado con que el admin pudiera perderse algún mensaje mientras atiende al anterior.

## Transiciones entre juegos

La función **set\_game(id, rounds)** del servidor, que se llama desde **set\_next\_game(id)** de game\_redirects.js solo se puede llamar después de que se haya llamado a la función **transition\_to\_next\_game().**

**transition\_to\_next\_game()** es obligatoria cuando se transiciona entre dos juegos (o cuando se elige el primero), por lo que es clave para ajustar cualquier valor que se quiera hacer cada ronda. Esta función se llama desde la homónima de JS, que a su vez se llama desde **admin\_next\_game().** Como se explicó antes, esta función devuelve si hay un juego listo o -1 si no, pero también hace todas las configuraciones necesarias para transicionar al siguiente juego (se haga inmediatamente en el caso de que haya uno configurado o más tarde desde el game\_selector.html). Estas funciones son:

* Incrementa el game\_number del servidor.
* Seguir el Schedule de las rondas configuradas. Si hay rondas configuradas del juego actual, resta una ronda, llama a set\_can\_players\_join(True) y devuelve el id del juego. Si no había, devuelve -1.

## Estructuras de datos y base de datos

Actualmente se manejan las siguientes estructuras de datos:

* Player:

|  |  |  |  |
| --- | --- | --- | --- |
| Variable | Tipo (Python) | Tipo (BBDD) | Descripción |
| id | int | int (PK) | ID único, autoincremental generado por la BD |
| name | str | varchar(20) (CK) | Nombre que se usa para el juego del ahorcado (impedir trolleos). Se utiliza para identificar al jugador para poder hacerle login, por lo que también debe de ser único. |
| nick | str | varchar(20) | Nombre artístico que se usa en todas las demás partes (admitir trolleos) |
| coins | int | int | Monedas que tiene el jugador en el momento |
| attributes | str | varchar(50) | Atributos de la persona separados por comas en el caso de que haya uno (para el juego del ahorcado) |
| elements | List of Any | No se guarda | Elementos que se guardan temporalmente durante uno de los juegos |
| logged | bool | No se guarda | Influye en si el juego espera al jugador antes de continuar |

* Prize

|  |  |  |  |
| --- | --- | --- | --- |
| Variable | Tipo (Python) | Tipo (BBDD) | Descripción |
| type | str | varchar(20) (PK) | Nombre del regalo (dulce, pequeño, mediano, grande) |
| prob | float | float [0-1] | Probabilidad de que toque (cambia automáticamente cuando no queda de alguno) |
| value | int | float [0-1] | Porcentaje de monedas que le cuesta al ganador (constante) |
| amount | int | int | Cantidad que queda del regalo (cada jugador trae 1, excepto en dulce, que se añaden 10 por jugador para que no se acaben) |

* Bet: apuestas de ruleta y tragaperras (no se guarda en BD)

|  |  |  |
| --- | --- | --- |
| Nombre variable | Tipo (Python) | Descripción |
| type | int/str | A qué se ha apostado |
| amount | int | Monedas que se han apostado |

Además, con la intención de obtener estadísticas de la partida, se van a añadir 2 tablas más:

Coins\_evolution: sirve para guardar las monedas que tenía cada jugador después de cada juego:

|  |  |  |
| --- | --- | --- |
| Nombre columna | Tipo (BBDD) | Descripción |
| id | int (FK y PK) | Referencia al jugador |
| game\_number | Int (PK) | Número del juego que se está jugando (cuántos se han jugado hasta ahora). Se genera desde Python mirando el mayor valor de esta tabla. |
| coins | int | Monedas que tiene el jugador en ese momento |
| year | int (PK) | Año en el que se está jugando la partida |

Prizes\_evolution:

|  |  |  |
| --- | --- | --- |
| Nombre columna | Tipo (BBDD) | Descripción |
| id | int (FK y PK) | Referencia al jugador |
| type | varchar(20) (FK y PK) | Referencia al regalo ganado |
| game\_number | int (PK) | Igual que la tabla anterior |
| year | int (PK) | Año en el que se está jugando la partida |