

Memoria Proyecto Final DAW

IES La SENIA |

Pintorrea

ruben saiz serrano

2020

Índice

1. **Introducción2**
2. **Idea de negocio4**
3. **Proyectos existentes2**
4. **Elección de las tecnologías2**
   1. **Backend** **2**
   2. **Frontend 2**
5. **Estructura de la aplicación2**
6. **Lógica del juego2**
7. **Introducción**

Este proyecto se trata de una web en la que su producto principal es un juego de dibujar y adivinar en el que hay dos tipos de roles:

* El encargado de dibujar la palabra que la web le sugiera.
* El encargado de adivinar que está dibujando el dibujante.

Los usuarios pueden registrarse e iniciar sesión con su cuenta o bien también pueden iniciar sesión con Google y a su vez serán registrados con los datos extraídos de cuenta de Google.

Para poder jugar con los demás usuarios debes entrar en una de las salas disponibles o bien crear una sala y compartir o bien el enlace completo para acceder a esa sala o compartir solo el código de la sala.

Todos los jugadores contendrán unas estadísticas con lo que ellos podrán diferenciarse de los demás jugadores en un ranking.

**2. Idea de negocio**

Recuerdo hace unos años cuando jugué por primera vez a este tipo de juegos (concretamente fue **Pinturillo 2**) con mis amigos que me gusto mucho y aun hoy en día solemos jugar de vez en cuando. Por aquel momento yo no sabía programar por lo que solo en pensar como realmente este tipo de juegos funcionaban por debajo se me escapaba.

Solo hace unos meses cuando tuvimos que decidir que tipo de proyecto íbamos a desarrollar, que por cierto yo no tenía ni idea de que iba a hacer, pero si que sabía que no quería hacer nada aburrido. Yo quería hacer algo que me motivará a seguir el desarrollo pasada la presentación del proyecto. Así que sin tener ni la mas remota idea de como hacer un juego online, decidí que quería hacer un ‘**Pinturillo**’.

**Que voy a ofrecer.**

Yo he sido consumidor de este estilo de juegos y se de que pie cojea cada uno de ellos, por ejemplo, **Pinturillo 2** no tiene cuentas de usuario, ni tiene personalización de avatares y tampoco ofrece clasificaciones.

Existen otros que, si que tienen todas estas carencias que **Pinturillo 2** tiene, pero no paginas tan famosas, yo pienso que es porque **Pinturillo 2** fue uno de los primeros en aterrizar en este mercado. Sea como sea Pintorrea cubrirá todos estos defectos y ofrecerá esta serie de puntos:

* Alta de usuarios.
* Acceso al juego con cuentas de Google.
* Clasificaciones.
* Personalización de avatares y subida de imágenes.
* Creación de salas privadas.
* Magnifica jugabilidad, con distintos tipos de lápices y colores.

**¿A quién va dirigido?**

Pienso que mi target principal es gente joven que normalmente juegue a videojuegos ya que con toda la gente que he jugado este juego siempre cumplían estas características. También existe un gran publico en las redes sociales como Youtube y Twitch gracias a creadores de contenido digital y ‘*streamers’*.

**3. Proyectos existentes**

Este tipo de juegos son muy populares debido a que todos ellos preceden del famoso [Pictonary](https://es.wikipedia.org/wiki/Pictionary) que fue creado por Rob Angel. Este juego consiste en adivinar una palabra a través de un dibujo hecho en papel.

Gracias a que la web fue creciendo y empezó a ser más y más popular entre la gente de calle, hubo gente que aprovecho la idea del Pictionary y creo juegos muy parecidos a él pero esta vez a través de una web.

Esto paso solo cuando la tecnología de la época empezó a permitir el desarrollo y esto fue entre 2010 y 2015, cuando los navegadores modernos como Firefox, Opera, Google Chrome y otros más empezaron a implementar los Web Sockets, esta tecnología permitió a los desarrolladores a crear aplicaciones con interacciones en directo.

Uno de los juegos más famosos, conocido por la mayoría de los adolescentes en España es [Pinturillo2](https://www.pinturillo2.com/), este fue uno de los primeros en ser viral en la web. Muchos “youtubers” y creadores de contenido digital compartían este juego por las redes.



Este juego cubre mínimamente las bases de Pictionary ya que no tiene funcionalidad extra.

Aquí es donde entra Pintorrea cubriendo los huecos de Pinturillo 2 con:

* Cuentas de usuario.
* Personalización del perfil; avatares, imágenes.
* Rankings.

**4. Elección de las tecnologías**

En este proyecto he querido mantener una consistencia a la hora de elegir los lenguajes de programación que iba a utilizar, quería que fuera robusto y rápido por eso he optado por utilizar nada más que Javascript tanto del lado del front como del back.

Se que existen muchos entornos de programación como pueden ser LAMP que se compone de Linux, Apache, MYSQL y Php, pero en mi caso veo mucho más limpio de mantener el código cuando está en el mismo lenguaje y también por supuesto nos quitamos de encima cualquier dolor de cabeza en las posibles futuras comunicaciones entre las distintas api’s de cada lenguaje.

**4.1 BACKEND**

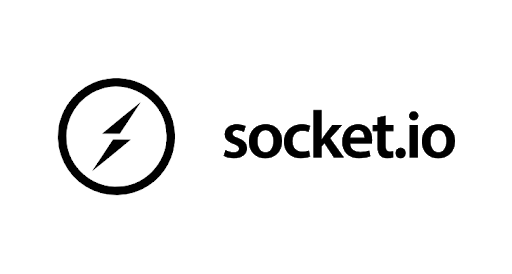
**NODE JS + EXPRESS**

Node js es un entorno en tiempo de ejecución multiplataforma basado en JavaScript, es asíncrono, su arquitectura es orientada a eventos y esta basado en el motor V8 de Google. Estos últimos años se ha hecho muy popular y empresas como Netflix o la NASA lo utilizan. En comparación a típicos servidores web como puede ser Apache este no tiene limite de peticiones ni hilos ya que ocurre todo en el mismo hilo.

Node js tiene un gestor de paquetes llamado NPM(node package manager), es de código libre y gracias a el puedes gestionar las dependencias de un proyecto.

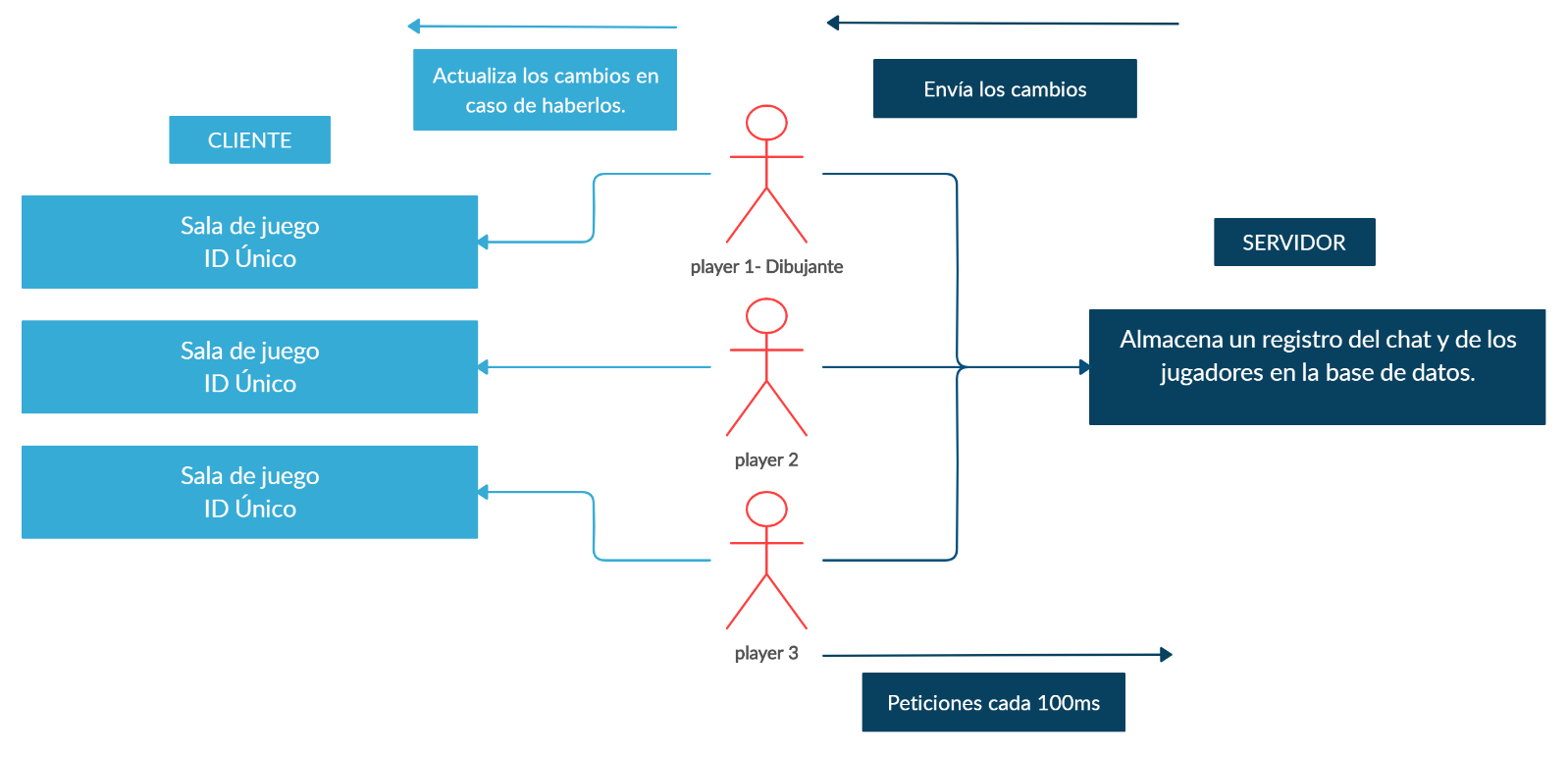
Express es un framework de Node js para crear con menos código y mucho más rápido aplicaciones web, te proporciona todo lo necesario; middlewares, rutas, motores de plantilla…

**SOCKET IO**



Una de las dificultades que he tenido con este proyecto ha sido la planificación de como hacer que la aplicación interactúe en tiempo real con todos los usuarios conectados en una misma sala.

Estas son dos de las ideas que he tenido para desarrollar esta funcionalidad:

La primera opción es **HTTP/REST.** Hacer una transferencia de datos por Ajax en la que cada uno de los clientes estará haciendo peticiones al servidor durante cortos plazos de tiempo. El cliente le hará llegar sus cambios, y el servidor después de almacenarlos le devolverá todos los cambios que hay en ese momento.

Esto es extremadamente pesado tanto para el servidor como para el cliente ya que con una petición cada 100 ms en una partida que puede durar por ejemplo 10 minutos estamos hablando de 6000 peticiones en una sola partida. Hay que decir que en la mayoría de estas peticiones no hay ningún cambio ni en el servidor ni en ninguno de los clientes.

Esto también requiere una lógica extra ya que tienes que evitar duplicidades en las respuestas a los clientes.

La segunda opción es con la ayuda de **WEB SOCKETS**, que, según la Wikipedia, “*es una tecnología que proporciona un canal de comunicación bidireccional y full-duplex sobre un único****socket****TCP. Está diseñada para ser implementada en navegadores y servidores****web****, pero puede utilizarse por cualquier aplicación cliente/servidor*”.

Gracias a que tiene una comunicación bidireccional no nos hace falta en el cliente estar preguntando constantemente al servidor por los cambios, sino que establecemos una conexión a un canal con el servidor y en cuanto haya cambios se dispara un evento.

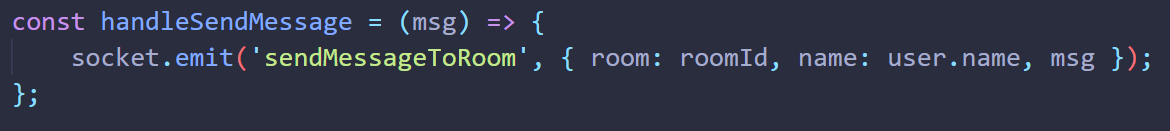
Ahora el cliente y el servidor solo accederían a los datos solo cuando los necesiten, pues bien, es evidente que para resolver este problema he escogido la segunda opción.

Pero esto no termina aquí ya que implementar web sockets de manera nativa es bastante complicado, así que después de una investigación por internet me he decantado por una librería llamada **socket.io**. Las razones son que es bastante fácil de utilizar y no necesitas tanto código como de manera nativa.

La transferencia de información en directo funciona mediante eventos los cuales son:

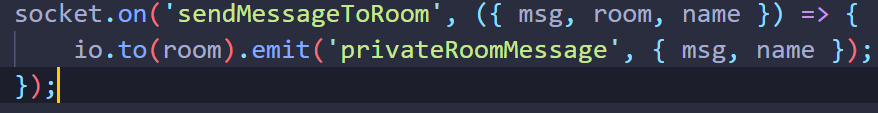
* Emit, comunicar a todo aquel que este escuchando.
* Broadcast, comunicar a todos menos al que está emitiendo.
* On, estar a la escucha de eventos.

Los clientes necesitan al servidor para poder comunicar al resto de clientes.

**CLIENTE EMITIENDO**

El cliente emite al servidor con los datos que necesite hacer llegar.

**SERVIDOR ESCUCHANDO Y EMITIENDO**

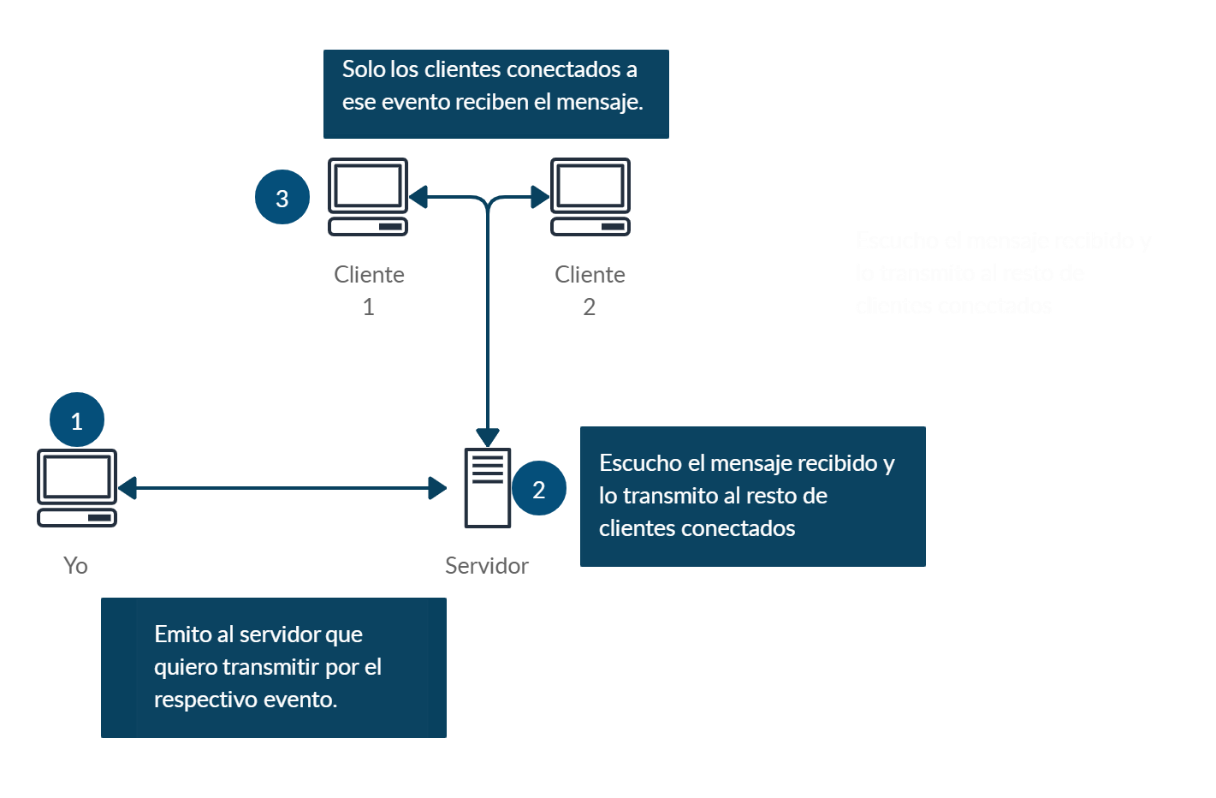
En este ejemplo el servidor crea un evento de escucha en cual su identificador es ‘sendMessageToRoom’ y al recibir la información lo que hace es transmitir a todo aquel que este escuchando en una sala con el identificador de evento ‘privateRoomMessage’.

**CLIENTE ESCUCHANDO**

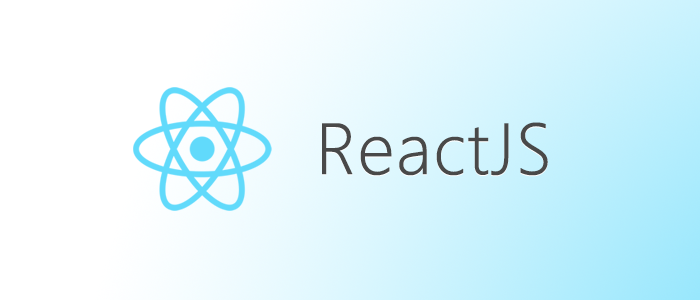


Aquí el cliente escucha cuando el servidor le emite un dato y lo que hace es invocar la función ‘sendMessage’.

Esto es un ejemplo básico de como funcionaria la comunicación en tiempo real entre los clientes y el servidor.



**4.1 Frontend**

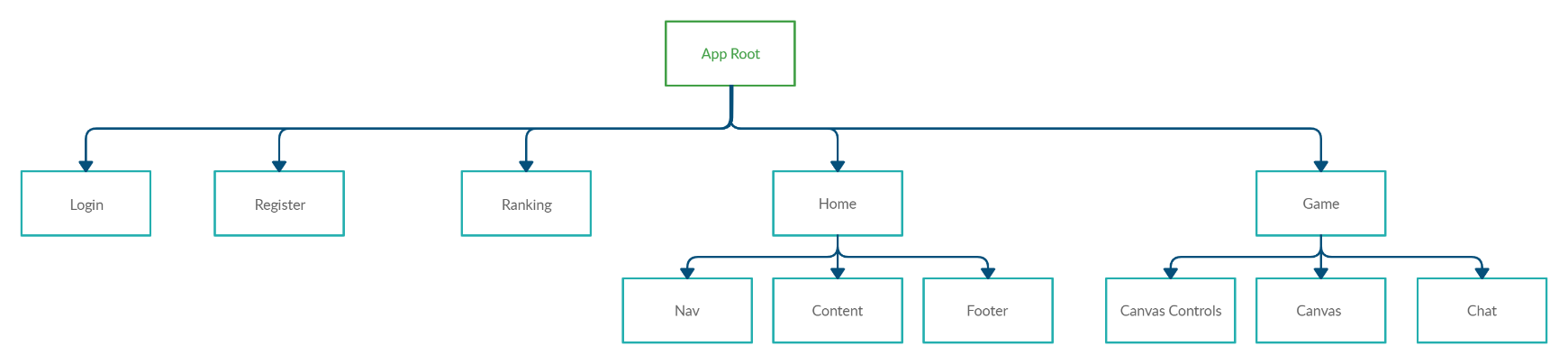
**REACT**

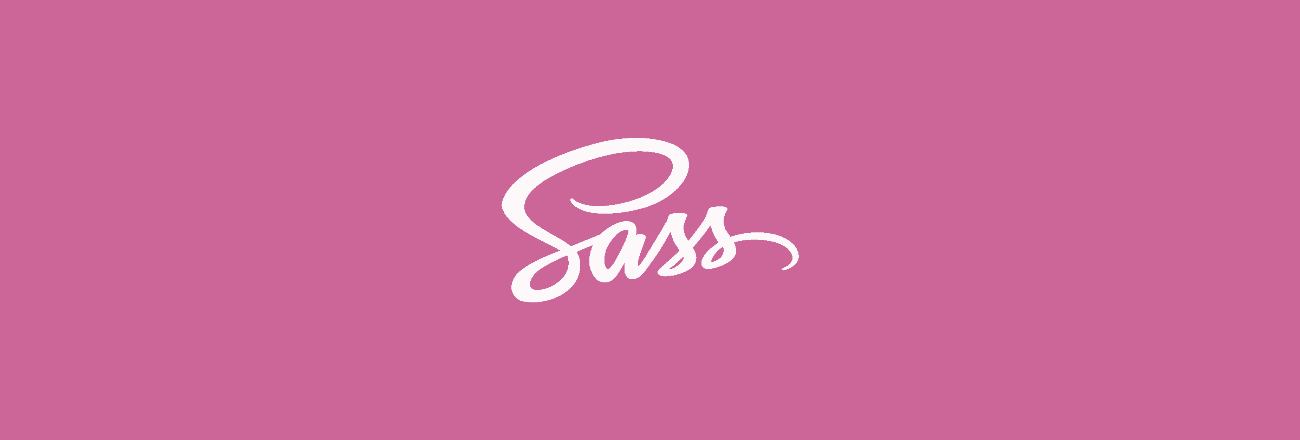
En el frontend he optado por utilizar principalmente **React**, es un framework desarrollado por Facebook. Se encarga de darle interactividad al DOM de una manera más eficiente que JavaScript puro ya que se apoya de lo que ellos llaman “Virtual Dom”, gracias a esto puede cambiar el estado de los nodos del DOM con menos manipulaciones.

Ya había utilizado recientemente así que no ha sido muy difícil decantarme por esta tecnología porque en mi opinión es mejor que JavaScript puro, estos son mis principales argumentos del porqué.

* Es de Facebook, es open source y Facebook lo utiliza en sus sistemas. Gracias a esto hay una gran comunidad de desarrolladores actualmente utilizando React.
* Utiliza el JavaScript más nuevo, últimas versiones de EcmaSript. Gracias a esto no caes en un estado de confort y aprendes cosas nuevas.
* Lenguaje JSX. Aunque tenga una apariencia a HTML esta es una extensión de JavaScript para escribir de una forma más elegante los componentes renderizados por el DOM.



La jerarquía de componentes a alto nivel de esta aplicación es la siguiente:

**SASS**

Sass es un preprocesador de estilos que trabaja con una sintaxis bastante parecida y te permite utilizar controles de flujo, bucles, funciones… Esto es una gran ventaja respecto a css porque gracias a esto me ha ayudado a recortar duplicidades en mi código y esto a la larga se agradece porque introducir cambios me será mucho más sencillo.

**ANT DESIGN**

Ant Design es una librería para React js la cual te provee componentes ya estilizados, como pueden ser botones, inputs, forms… Esto te ayuda a no tener que estar preocupándote de que tus estilos se ven bien en diferentes navegadores ya que ellos ya se encargan de esto.

**REDUX**

Redux un contenedor del estado de una aplicación JavaScript, te ayuda a escribir aplicaciones que se comportan de manera consistente. En las aplicaciones de React, Redux evita tener que estar propagando estados entre componentes.

Al tener un único contenedor del estado de la aplicación puedes acceder a él desde cualquier lugar.

Esto también te ayuda a desacoplar el estado global de una aplicación frontend de la parte visual ya que si tuvieras que hacer una petición a una api Redux sería el encargado.

Redux tiene 3 conceptos importantes y son:

* **Única fuente de la verdad**: En Redux hay un único objeto que almacena el estado de toda la aplicación.
* **Inmutabilidad, el estado es *read-only:*** Ninguna interacción puede cambiarlo directamente. Lo único que puedes hacer para conseguirlo es emitir una acción que expresa su intención de cambiarlo.
* **Funciones puras**: Usa funciones puras (a mismos inputs, mismos outputs) para definir como cambia el estado en base a una acción. En Redux estas funciones se conocen como reducers y al ser puras, su comportamiento es predecible.

**REDUX THUNK**

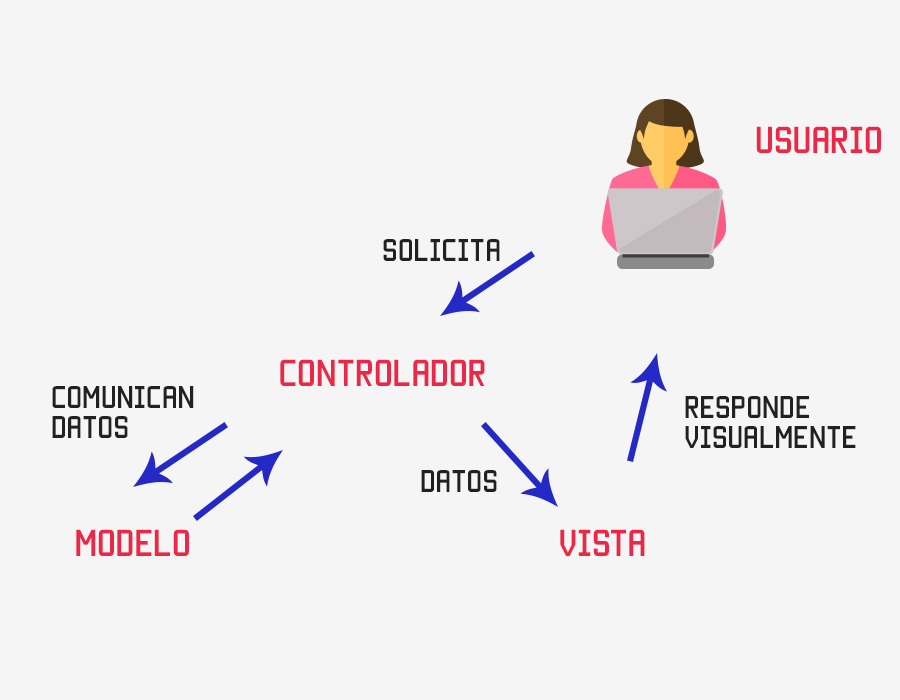
Redux Thunk es un middleware para hacer peticiones asíncronas a través de los actions de Redux.

**4. Estructura de aplicación**

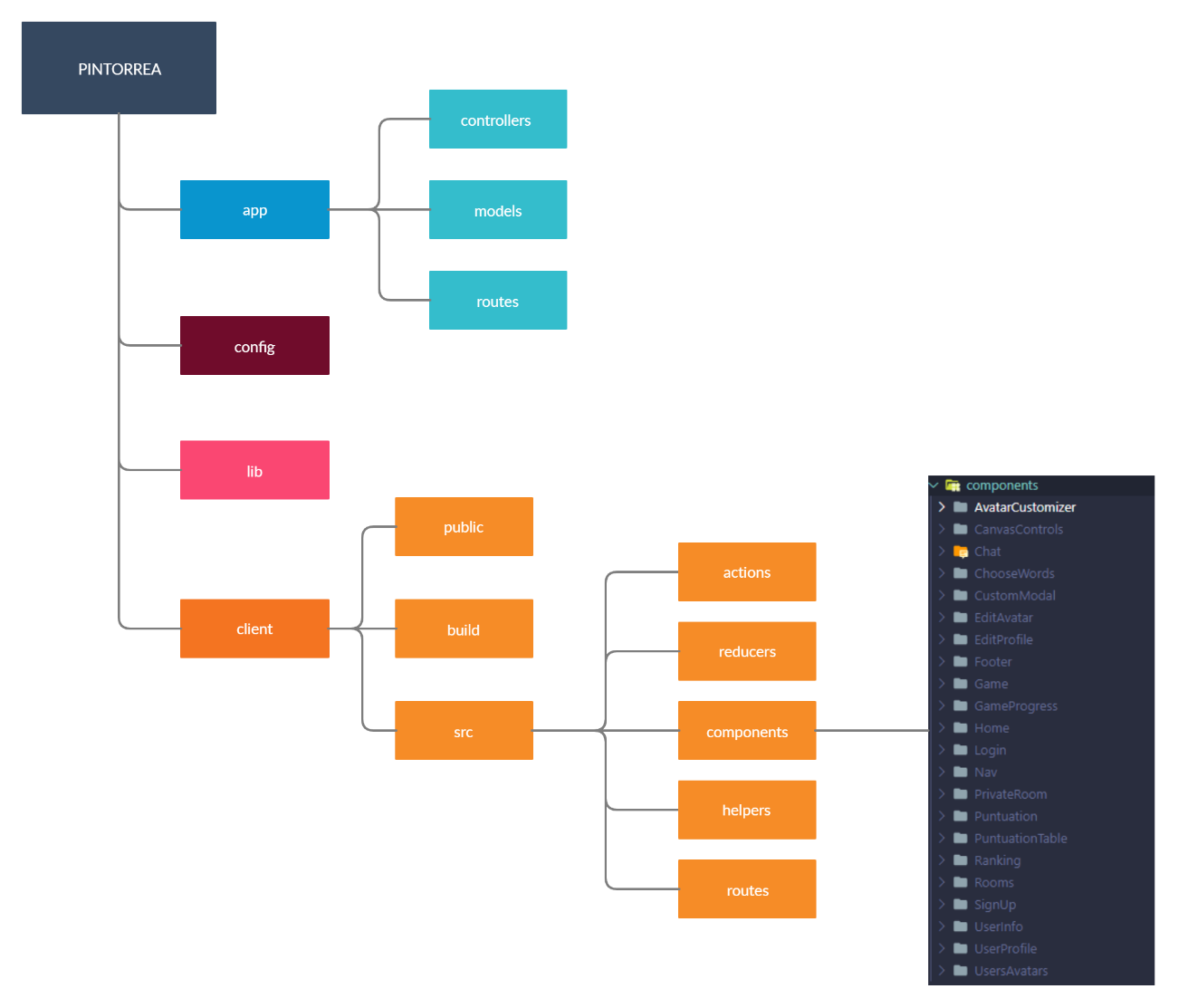
Para empezar con buenas practicas el proyecto lo primero que me he planteado es que patrón de diseño implementar, esto me permite tener una estructura del proyecto mas entendible.

En arquitectura del software existen patrones de diseño que son soluciones generales a un problema común dentro de un contexto. Yo me he decantado por el patrón MVC (modelo, vista, controlador) porque este te divide la aplicación en 3 partes:

* Modelo: es quien accede a los datos, ya sea base de datos o una api.
* Vista: representación visual de los datos.
* Controlador: encargado de resolver eventos de la vista y responderle con los datos extraídos del modelo.



Después de la elección del patrón de diseño ya puedo realizar la creación de carpetas.



Todo el apartado de app es el modelo-vista-controlador gestionado por el servidor, hay un extra que sería la carpeta routes, esta carpeta contiene el punto de entrada a nuestra aplicación mediante endpoints a través de los verbos Http (get, post, put, delete, patch…).

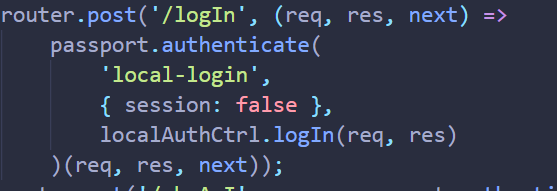
**AUTENTICACION Y AUTORIZACION**

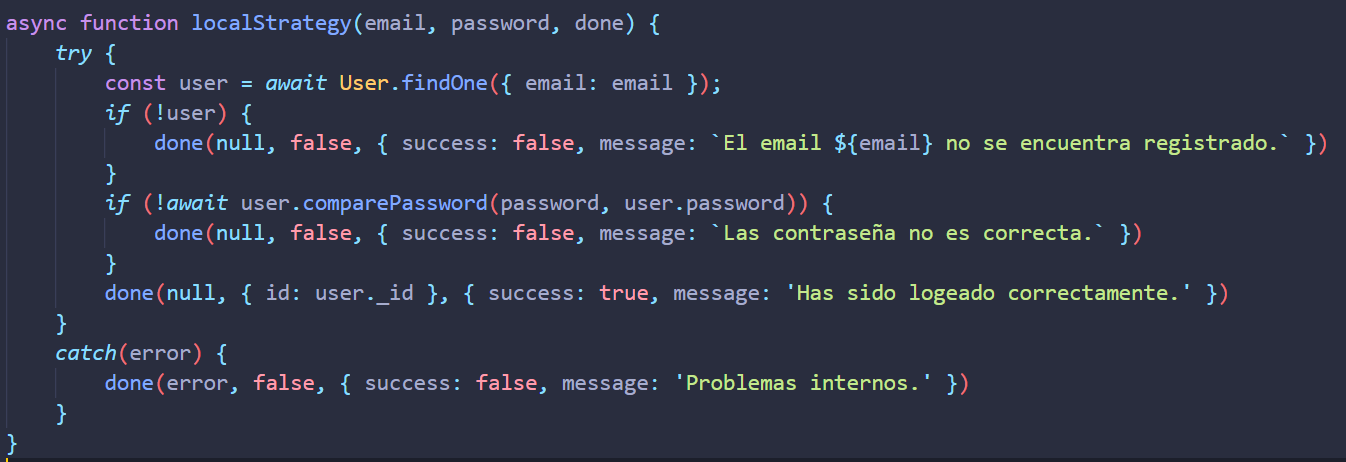
Para la autenticación utilizo [JSON WEB TOKENS](https://es.wikipedia.org/wiki/JSON_Web_Token) que en la práctica son una cadena de texto en base 64, cada una de ellas esta separada por un punto y significan:

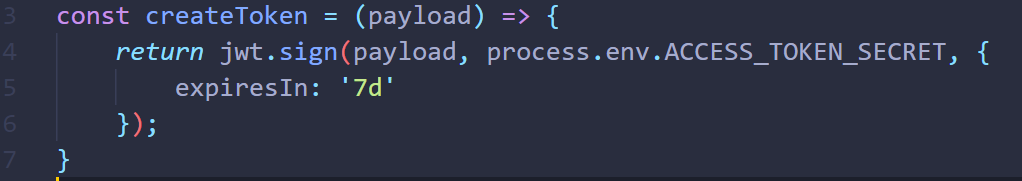
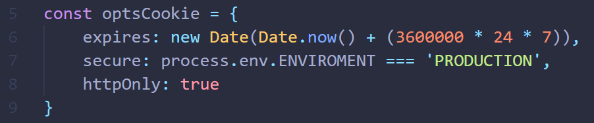
* **Header**: encabezado donde se indica, al menos, el algoritmo y el tipo de token.
* **Payload**: donde aparecen los datos del usuario, yo en particular solo almaceno el id del usuario de la base de datos.
* **Signature**: esta firma nos permite verificar si el token es valido o no lo es.

Este token es codificado con una cadena de texto que nosotros debemos de almacenar y no compartirla por ningún lado. Una vez codificado, el contenido del token es visible para cualquiera ya que el token no esta cifrado, sino que esta codificado. Pero el quid de todo esto es que podemos saber si en el trayecto de cliente-servidor el token ha sido modificado

Una vez explicado que es un JSON WEB TOKEN, procedo a explicar cómo gestiono mi autenticación y autorización de usuarios.

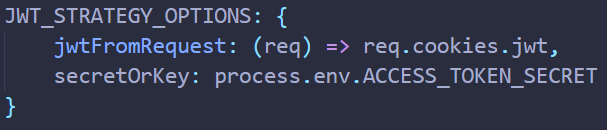
Cuando el usuario envía las credenciales al servidor compruebo que estos son realmente válidos. Toda esta validación la hago con Passport js:

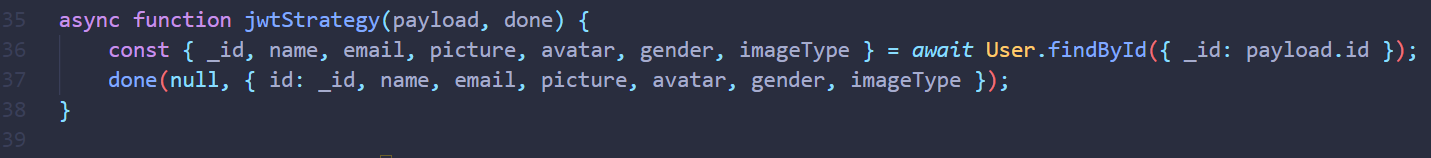
Este es el endpoint por donde entrarían las credenciales, y este llama a la estrategia de Passport ‘local-login’ que yo personalmente he configurado.

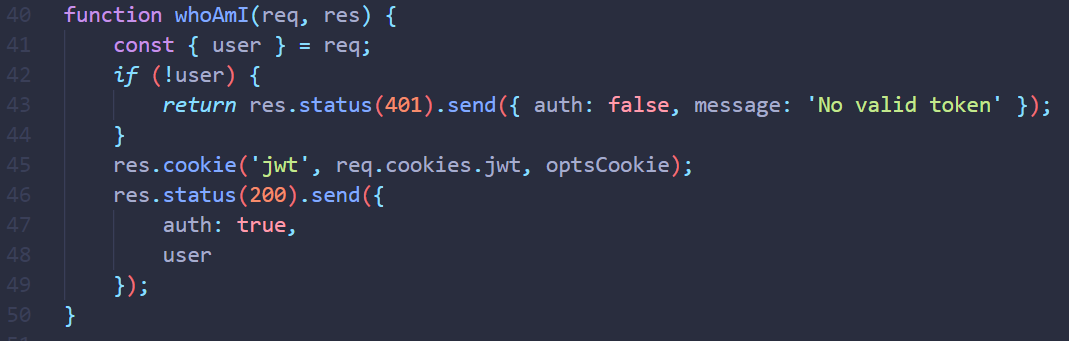
Si el email y el password son correctos entonces ya pasamos a generar el token con su id de la base de datos y enviarlo al cliente.

Ahora cuando el cliente necesita algún recurso o bien intenta acceder a alguna parte del cliente tiene que pasar por este middleware:

Este invoca a otra estrategia de Passport en la cual lo que se hace es extraer el token de las cabeceras y validarlo:

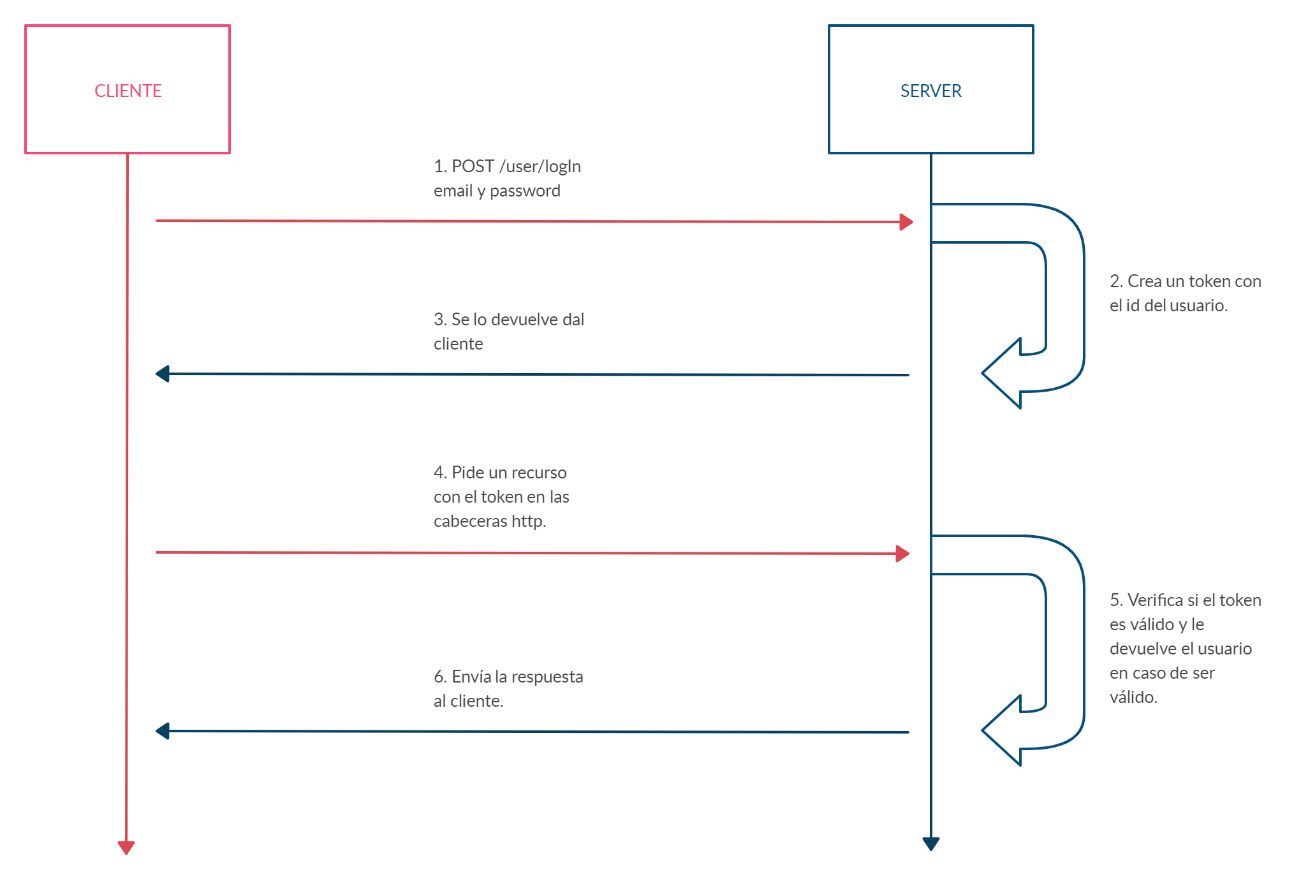




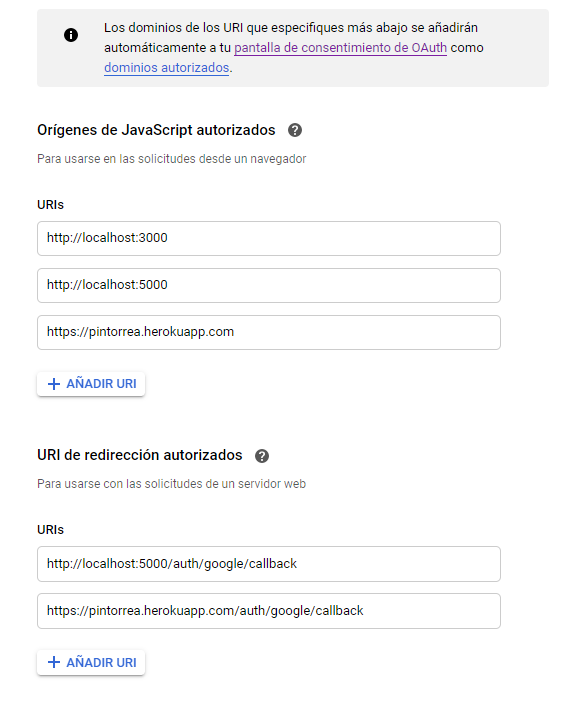
Llegados a este punto ya accedemos a la base de datos y accedemos a la información que necesitemos del usuario.

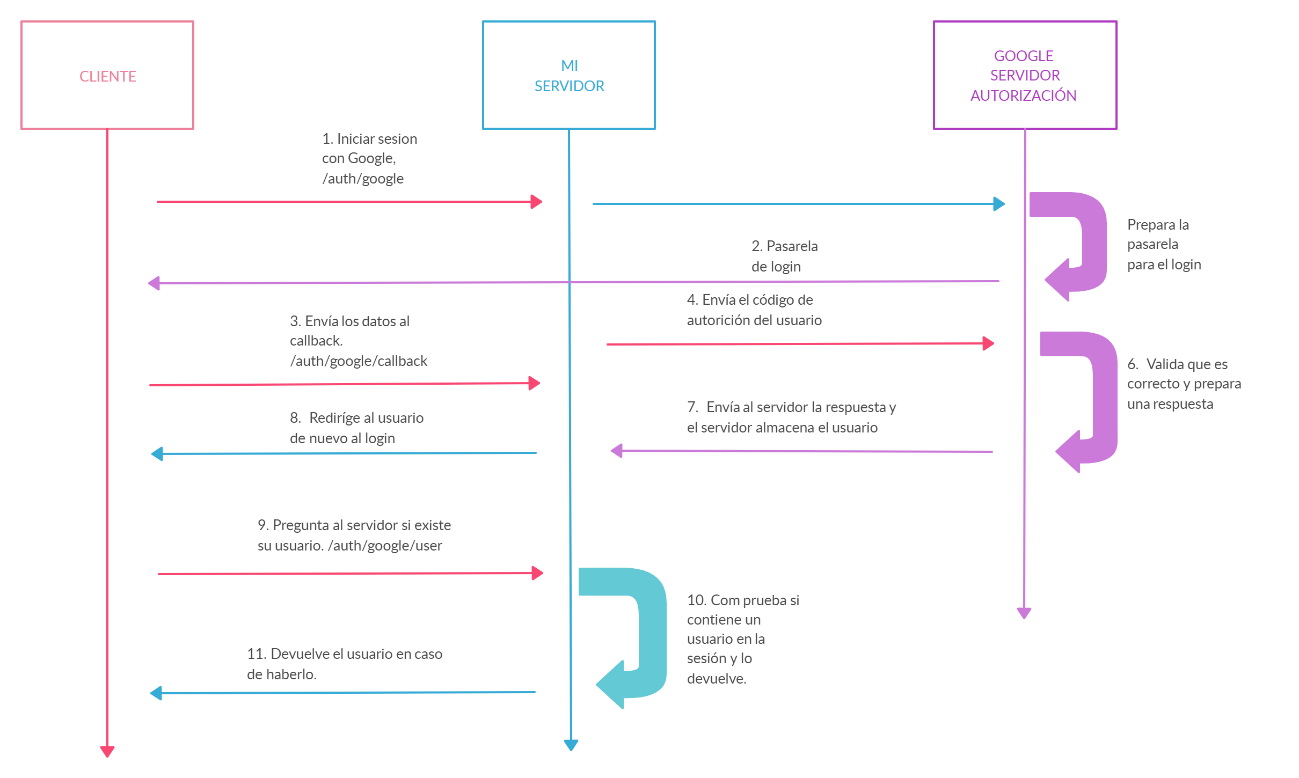
Regeneramos la cookie con los datos del usuario y se lo mandamos.

Con esta implementación nos estamos ahorrando una consulta a la base de datos cada vez que el cliente requiera algún recurso. Con json web tokens solo consultamos la base de datos una vez que el token es válido. Esto nos abre las puertas a la escalabilidad porque en el caso de necesitar por ejemplo una aplicación móvil podrían autenticarse con esta estrategia, en cambio con la implementación de sesiones esto sería un problema ya que los móviles no tienen sesiones ni cookies como los navegadores web.



Mi aplicación también permite acceder a través de Google, este tipo de atenticaciones se suelen hacer con un estándar llamado OAUTH 2.0 el cual delega la autenticación en aplicaciones de terceros. Google implementa este patrón, requiere que te crees un cliente api con unas credenciales. Esto lo hacen para ellos saber quien va a estar preguntando por dicha información de x usuario.

En el panel que nos ofrecen ellos saben los dominios autorizados y las redirecciones también. Con esto ellos se aseguran de que ningún otro dominio que no este inscrito aquí pueda recoger ningún tipo de información.



Este sería el flujo de mi aplicación cuando se intenta acceder con Google.

**6. Lógica del juego**

La lógica de mi juego es bastante sencilla:

* Las rondas duran 80 segundos (salvo en las privadas).
* Hay 3 rondas, para completarse una ronda tienen que haber dibujado todos los de la sala.
* El dibujante puntúa cuando alguien acierta lo que esta dibujando, suma el tiempo actual de la sala dividido por el resto de jugadores de la sala.
* Los no dibujantes al acertar puntúa el tiempo actual de la sala.
* Si todos los jugadores de la sala han acertado, la sala parará y mostrará los resultados del turno.
* Al llegar al limite de rondas se mostrará el ranking de la partida ordenada descendentemente por puntuación.

**7. Usabilidad**

He tenido bastante en cuenta las reglas de usabilidad