GRADO EN INGENIERÍA DEL SOFTWARE

Escuela Técnica Superior de Ingeniería de Informática

Curso académico 2022-2023

Trabajo fin de grado

Pizarra Colaborativa en Angular

Tutor: Micael Gallego Carrillo

Autor: Miguel Rodriguez Àlvarez

Agradecimientos

Me gustaría agradecer a mi profesor y tutor del TFG Micael Gallego a el apoyo y la confianza depositada en mí para realizar este proyecto puesto que me lo propuso y me ha guiado a lo largo de su desarrollo.

También me gustaría agradecer el cariño de mis padres y mi hermana que me han acompañado todos estos meses preocupándose por mi y apoyándome.

Por último pero no menos importante, agradecer a mis amigos, y sobretodo mi novia, por el apoyo, los ánimos que me han dado y sobretodo la paciencia de verme encerrado más de lo que les gustaría para terminar el proyecto.

Madrid, 28 de junio de 2023

Miguel Rodriguez Àlvarez

Resumen

En este documento voy a explicar de forma detallada en qué ha consistido mi Trabajo de Fin de Grado, el desarrollo de una pizarra colaborativa desarrollada en Angular.

Para ello, voy a dividir el documento de manera que, en primer lugar, haré una breve introducción sobre el contexto y motivación que me ha llevado a realizar este proyecto. Tal y como voy a contar, la situación en la que hemos vivido estos tres últimos años ha condicionado la forma en la que trabajamos o damos clases.

En segundo lugar, voy a explicar los objetivos que he querido lograr y que me han motivado a crear esta pizarra. Ha sido un trabajo largo y costoso y tener claro estos motivos me ha ayudado a continuar para poder finalizarlo.

En el tercer punto explicaré de forma global cada una de las tecnologías, herramientas y metodologías que he utilizado a lo largo del proyecto, intentando ser lo más claro posible para que en los siguientes puntos se pueda entender sin problema el funcionamiento de la pizarra.

En el cuarto punto desarrollaré el diseño y la arquitectura utilizada, explicaré los puntos claves del código y las pruebas realizadas. Me apoyaré de distintos diagramas, figuras y en ocasiones fragmentos de código para intentar que se entienda de la mejor manera posible la parte más técnica del proyecto.

Para finalizar, voy a hablar de mis conclusiones finales tras acabar el proyecto junto a las limitaciones que puede tener, los problemas que he tenido y que no he podido solventar y se podrían tener en cuenta para trabajos futuros, ya que la pizarra es una buena herramienta con mucho potencial.

Índice general

Índice de figuras

Listado de códigos

Capítulo 1
Introducción y Motivación

Este proyecto consiste en la realización de una pizarra colaborativa en Angular de manera que dos o más personas puedan dibujar, representar ideas o realizar diagramas de forma online a tiempo real. Cada usuario puede tanto insertar nuevos elementos como editar los que otros usuarios hayan incorporado.

Para entender cómo llegué a la conclusión de que esto podría ser una buena idea como proyecto, debemos echar la vista atrás en el tiempo, concretamente en el año 2020. En Marzo de este año tuvimos que enfrentarnos a la llegada de la pandemia causada por el virus Covid-19, algo para lo que no estábamos preparados y que nos obligó a cambiar por completo muchos aspectos de nuestra vida cotidiana. Entre ellos, nuestras relaciones interpersonales.

Una pandemia que aún hoy en día seguimos enfrentándonos, aunque, afortunadamente, gracias a los esfuerzos tecnológicos, médicos y humanitarios con unas consecuencias mucho más leves.

Esta pandemia fue causada por el virus SARS-CoV-2, el cual se transmite mediante pequeñas gotas de saliva que se emiten al hablar, estornudar, toser o respirar. No es necesario que sea contacto directo, ya que estas partículas pueden estar suspendidas en el aire de un espacio cerrado e inhalarlas de forma indirecta, o que por contacto directo toquemos algún objeto con partículas y después nos llevemos la mano a la boca.

A día de hoy, seguimos sufriendo sus consecuencias llegando a cifras muy elevadas de infectados por día, aunque afortunadamente los síntomas generados por este virus son más leves que en sus inicios. Esto es debido a los grandes avances producidos por los sanitarios y la tecnología con las vacunas y a las numerosas mutaciones que ha ido

teniendo el propio virus.

El medio de contagio es el aire, pero esto no se sabía a ciencia cierta al comienzo de la pandemia. Teníamos muy poca información acerca del virus y sólo sabíamos lo que ya ocurría en otros países como China o Italia.

Por ello la principal medida que se tomó fue la del confinamiento. Un confinamiento estricto en casa que implicaba no ir al trabajo, no ir a clase e incluso algo tan cotidiano como hacer la compra tuvo que limitarse yendo un único miembro de cada familia.

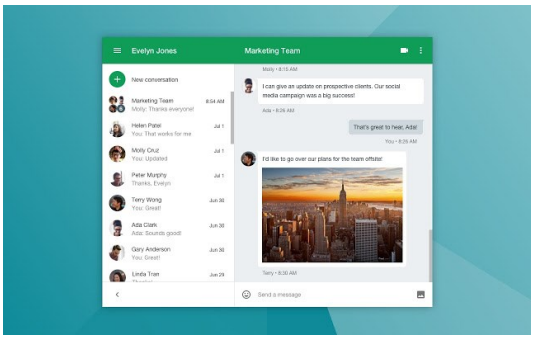
Tanto en el mundo laboral como en el académico, hizo que tuviéramos que replantearnos nuestra forma de trabajar y examinarnos. No podíamos asistir a la universidad o a la oficina, por lo que comenzamos a usar herramientas digitales que nos permitiesen desempeñar todas las funciones que hacíamos anteriormente pero desde nuestra casa.

Esta situación hizo que en numerosas ocasiones echemos en falta, principalmente en el ámbito académico, la presencialidad a la hora de dar una clase o tener una reunión. Al estar en persona, estamos más dispuestos a prestar mejor atención y a su vez somos capaces de explicar mejor las ideas, temas o conceptos.

Para poder combatir con este problema, hemos intercambiado la presencialidad en la oficina por el teletrabajo, las tutorías entre el profesor y alumno en despacho ahora son desde casa y los exámenes donde nos juntábamos gran número de alumnos se han realizado de forma online desde el Aula virtual.

Por lo tanto, ¿cómo hemos sido capaces de adaptarnos a esta nueva situación teniendo que hacer todo desde casa?
Nos hemos visto en la obligación de utilizar nuevas herramientas. Nuevas maneras

(a) Teams



(b) Hangout

(c) BBCollaborate

Figura 1.1: Herramientos de videollamada

de poder conectarnos entre nosotros para poder realizar aquellas tareas que hacíamos presencialmente.
Ahora las reuniones no son en la oficina, ni las clases son en los aulas, desde Marzo del 2020, usamos aplicaciones de videoconferencias, llamadas en las que nos conectamos varias personas para llevar a cabo aquello que se haría en la oficina o en un aula. Tal y como podemos ver en la figura 1.1, las aplicaciones que todos conocemos (a) Microsoft Teams, (b) Google Hangout o (c) BlackBoard Collaborate son algunas de las muchas que nos hemos acostumbrado a usar a diario.

Por ello, considero que desarrollar una pizarra colaborativa, desarrollada con Angular y siendo una tecnología muy potente y actual, puede ser una gran idea para que muchas empresas, profesores o incluso cualquier persona que esté desarrollando una aplicación web, la incorporen de manera gratuita simplemente añadiendo este componente a sus aplicaciones.

Según una encuesta realizada por Adriana Scozzafava, Ejecutiva Senior del IESE Business School y presentada junto a la directora general de la Cámara de España, Inmaculada Riera; la presidenta de Woman In a Legal World, Marlen Estévez y la

presidenta de 50&50 Gender Leadership, Gloria Lomana, 'el 74 % de los españoles considera que su trabajo le permite trabajar en remoto total o parcialmente'.

Esta encuesta la podemos encontrar en la página web de la Cámara de Comercio de España [Scozzafava, 2020] y nos da una visión sobre la opinión de muchos de los españoles en relación al teletrabajo. Sobre todo en los empleos que requiere de un ordenador para trabajar, puesto que, a día de hoy, muchos de ellos están digitalizados y no requieren en gran parte la presencia en oficina.

Existe, aún así, una discusión en cuanto a la productividad de los empleados trabajando desde casa comparándola con la producida desde la oficina. Puesto que, por un lado se puede tener un mayor control sobre los empleados, pero a su vez, desde sus casas, los empleados se sienten más libres y más eficientes al disponer más tiempo a lo largo del día.

A raíz de este tema, considero importante el desarrollo de herramientas digitales como es el caso de una pizarra colaborativa con la que poder lograr una mayor virtualización de lo que es una clase o reunión de trabajo presencial pero hecha desde casa.

De esta forma este proyecto sirve para completar las herramientas de videoconferencia que utilizamos para comunicarnos en una reunión o clase que nos permite además representar aquellas ideas y conceptos difíciles de transmitir verbalmente o conceptos complicados que con una pizarra digital podemos ayudarnos y entendernos mejor.

La motivación más relevante que me ha llevado a desarrollar este proyecto es que existan aún más posibilidades a la hora de utilizar herramientas digitales, tanto en el ámbito académico como en el profesional, o incluso en aplicaciones web de entretenimiento.

La búsqueda y mejora de nuevas librerías que nos permitan incorporar esta tecnología a otras aplicaciones es otra de las finalidades de este proyecto. Es importante en el mundo del desarrollo que tengamos acecso a código libre para incorporar nuevas funcionalidades de otras personas, o incluso poder mejorar las existentes para ampliar nuestos conocimientos.

Tal y como estamos viendo actualmente, nos vemos obligados a realizar más tareas online, desde cualquier punto sin necesidad de estar presencialmecnte. El desarrollo de esta pizarra colaborativa en Angular que se pueda incorporar de forma gratuita a cualquier aplicación permite que existan más facilidades a la hora de comunicarse en una reunión o una clase.

Con esta pizarra mi intención es crear un componente Angular que la gente pueda incorporar a sus proyectos, a sus aplicaciones, para que sirva en proyectos de entretenimiento o juegos que requieran una pizarra, o proyectos para empresas que necesiten comunicarse varias personas y puedan dibujar a tiempo real, o para ámbito académico y que un profesor pueda explicar la arquitectura de un proyecto mientras sus alumnos plantean las dudas sobre la pizarra.

Capítulo 2
Objetivos

En este capítulo voy a detallar los distintos objetivos que quiero alcanzar con este proyecto.
Tal y como he contado en el capítulo anterior, venimos de una situación complicada en la que requerimos de mayor accesibilidad a plataformas digitales, aplicaciones web y móviles y herramientas tecnológicas que nos permitan llevar nuestro día a día cada vez más desde Internet.

La idea principal de este proyecto era conseguir una pizarra que cumpliese al menos las funcionalidades más básicas de una pizarra digital, a mano alzada, dibujar formas simples o introducir texto. Sin embargo, a medida que los requisitos principales se veían realizados, pude ampliar más funcionalidades para lograr una pizarra más completa.

Otro de los puntos que quería lograr en este proyecto era realizar la pizarra con tecnologías modernas, como es Angular o Socket.io, de manera que pudiera servirme, como para ampliar mis conocimientos en el área del desarrollo web, como para que la pizarra fuera fácilmente implementable en proyectos de terceros.

Por ello, el objetivo principal de este proyecto es dearrollar una pizarra colaborativa, de código abierto y con tecnologias sencillas y actuales que permitan concetar a 2 o más usuarios y dibujar de forma dinámica sobre ella.

Capítulo 3
Tecnologías, Herramientas y Metodologías

A continuación voy a explicar de forma detallada las tecnologías, herramientas y metodologías que he utilizado a lo largo de este proyecto.

Contaré desde las herramientas, los lenguajes de programación y los diferentes frameworks, hasta la metodología usada y las herramientas que me han ayudado a la realización del proyecto.

3.1. Herramientas

Como herramientas principales que he usado en el proyecto de la Pizarra podemos destacar Visual Studio Code y GitHub. Son indispensables en este proyecto debido a las posibilidades que nos ofrecen.

Visual Studio Code es el entorno de desarrollo utilizado mientras que GitHub es la herramienta con la que he podido tener guardado y sincronizado mi repositorio.

3.1.1. Visual Studio Code

Visual Studio Code es el editor de texto que he elegido para desarrollar este proyecto.

Se trata de un editor de código fuente gratuito desarrollado por Microsoft para Windows, Linux y macOS muy completo y versátil, lo que lo convierte en una de las primeras opciones a la hora de desarrollar los proyectos para los programadores.

En mi caso, quise utilizar este editor de texto ya que me permite trabajar de una forma muy cómoda y sencilla gracias, entre otras cosas, a las funciones de IntelliSense o la integración de Git, del cual hablaré más adelante.

Las diferentes funciones de IntelliSense que ofrece Visual Studio Code nos permite tener un mayor conocimiento sobre las librerías que usamos, ya que nos permite conocer la información de los métodos, parámetros, propiedades de un objeto o incluso nos sugiere métodos a medida que codificamos.

En este proyecto he utilizado varias librerías de terceros en las que, gracias a estas funcionalidades, he podido entender y conocer mejor su funcionamiento y así poder desarrollar mejor y más rápido, puesto que no he necesitado dedicar tiempo a investigar qué método me convenía más o qué propiedades podía utilizar en cada momento.

Por otro lado, gracias a la integración de Git que ofrece Visual Studio Code, he podido tener un claro y seguro control de las versiones y actualizaciones en mi repositorio.

3.1.2. Github

GitHub [git, 2023] es una herramienta con la que podemos almacenar y gestionar nuestros proyectos utilizando el sistema de control de versiones de Git.

Con Git podemos tener un control de versiones de nuestro código de forma que podamos guardar el trabajo realizado o tener distintas versiones disponibles. De manera que podamos volver a una versión anterior de nuestro proyecto si lo necesitamos entre

otras muchas cosas.

Es una herramienta muy útil sobre todo para proyectos en los que se involucran varios programadores, ya que pueden organizar sus tareas creando ramas de trabajo independientes y unirlas una vez hayan finalizado.

En mi proyecto he utilizado GitHub porque me ha permitido clonar proyectos de pizarras con los que poder trabajar y usar como base, así como tener un control de mis versiones y las tareas que iba realizando.
Al haber desarrollado dos versiones distintas usando dos proyectos de pizarra digital distintas, he podido controlar correctamente el proceso realizado en cada una de las versiones.

3.2. Tecnologías

3.2.1. Nodejs

Node.js ([nod, 2023]) es un entorno en tiempo de ejecución para la capa del servidor. Fue creado por los desarrolladores originales de JavaScript y por ello es un entorno basado en este mismo lenguaje.

Se trata de un entorno que utiliza el modelo de entrada y salida sin bloqueo y permite que nuestras aplicaciones sean más ligeras y la comunicación entre sockets emitiendo y recibiendo eventos sea más eficiente.

Entre sus funcionalidades más útiles puedo destacar Nodemon, una herramienta que te ayuda a agilizar el inicio de tus aplicaciones y tiene la particularidad de reiniciarlas automáticamente al detectar si un fichero ha sido modificado. Me resultó muy útil ya que podía desarrollar mi servidor al mismo tiempo que se ejecutaba y así ver en tiempo real los cambios que implementaba.

Al estar ideado como un entorno de ejecución de JavaScript orientado a eventos asíncronos, consideré que era la mejor opción para mi proyecto de pizarra colaborativa donde utilizo Node.js junto a la librería Socket.io para comunicarnos entre usuarios que están en la misma sala para dibujar.

3.2.2. Angular

Angular ([ang, 2023]) es un framework de código abierto desarrollado por Google que nos permite crear aplicaciones web de una sola página, Single Page Application (SPA).

Las aplicaciones SPA, y en concreto las desarrolladas con Angular, nos permiten navegar por nuestra aplicación en una única página. Esto se debe a que la aplicación realmente carga una sola ruta y mediante componentes navegamos entre las diferentes páginas que queramos que tenga la aplicación.

Por ejemplo, en mi aplicación de demostración, cargamos la pagina principal y a la hora de navegar, realmente lo que estamos haciendo es cargar distintos componentes sobre esa ruta, como la sala principal o el propio componente del canvas. Con Angular, podemos crear diferentes componentes que se pueden ir creando, mostrando, ocultando o eliminando a medida que navegamos por la aplicación.

Estos componentes se crean en el lenguaje TypeScript, que define un conjunto de tipos JavaScript, lo que permite que podamos ejecutar nuestra aplicación desde cualquier plataforma.

Mediante una terminal desde nuestro proyecto, Angular nos ofrece una serie de comandos muy útiles que nos ayudan a compilar, actualizar, añadir componentes o lanzar la aplicación. Se suelen iniciar con las dos letras ‘ng’ y según la tarea que deseamos realizar, podemos crear un nuevo proyecto, generar nuevos componentes o incluso pedir ayuda en caso de que no sepamos qué comandos ofrece o para qué sirve

cada uno.

Angular ha sido el framework elegido para la realización de este proyecto, ya que se trata de una tecnología muy usada actualmente y que nos ofrece muchas ventajas en el mundo de la programación web.

3.3. Lenguajes

Debido a que estoy creando una pizarra colaborativa para que sea implementada en una aplicación web de Angular y debido a las tecnologías utilizadas, podemos imaginar que los dos lenguajes de programación que he usado han sido tanto TypeScript como JavaScript, ambos propios del desarrollo de aplicaciones y páginas web, así como el propio lenguaje de marcas HTML y CSS.

A continuación haré una breve explicación de qué se trata cada lenguaje y para qué se ha utilizado.

3.3.1. TypeScript y JavaScript

Una vez que ya he podido explicar a grandes rasgos qué es y cómo se usan Angular y Node.js, podemos entender el porqué de la utilización de estos dos lenguajes de programación en mi proyecto de la pizarra colaborativa.

JavaScript ([js, 2023]) es un lenguaje de programación multiplataforma, interpretado, y orientado a objetos. Fue desarrollado por Brendan Eich de Nestcape originalmente con el nombre de Mocha y después renombrado a LifeScript para finalmente acabar llamándose como hoy en día lo conocemos, JavaScript.

Su principal uso es en el desarrollo web, ya que puede ejecutarse en un navegador web haciendo que la experiencia del usuario en su paso por Internet sea más dinámica, atractiva y eficiente.

Por otro lado tenemos otro lenguaje muy similar, TypeScript ya que extiende del lenguaje de programación JavaSrcript, lo que nos permite tener un amplio abanico de posibilidades sobre todo en la programación web.

TypeScript ([ts, 2023]) fue desarrollado durante 2 años por Microsoft, y publicado en 2012. Hoy en día es junto a JavaScript los lenguajes más usados para la creación de páginas y aplicaciones web, al destacar tanto en la ejecución en navegadores web como en la parte del servidor.

Angular nos ofrece muchas ventajas al utilizar Typescript como lenguaje. Una de las ventajas más significantes que podemos encontrar es que es un lenguaje compilado. A la hora de tener que buscar errores en nuestro código, JavaScript en tiempo de ejecución es en ocasiones difícil. Sin embargo, con Typescript esto no es así, ya que nos permite encontrar errores en nuestro código en tiempo de compilación. Esto se traduce en ahorro de tiempo y eficiencia a la hora de desarrollar nuestro proyecto.

3.4. Librerías

Metiéndonos un poco más a fondo en la propia aplicación, podemos encontrar que he usado algunas librerías que me han ayudado con las distintas funcionalidades que quería implementar para cumplir con los requisitos principales; que sea una aplicación colaborativa y que tuviese las funcionalidades básicas que puede tener una pizarra digital normal. Entre las librerías utilizadas encontramos ng-whiteboard-canvas, Fabric.js, Socket.io, Color Picker Angular, Jest y Selenium. Haré una breve explicación de en qué consiste cada librería, para qué me ha servido y los principales problemas y ventajas que he tenido con ella.

3.4.1. Ng2-whiteboard-canvas

La idea inicial de este proyecto era realizar una pizarra a partir del proyecto Ng2-Whiteboard-Canvas completando las funcionalidades que consideramos que le faltaban así como la implementar el requisito de ser una pizarra colaborativa online donde más de una persona pudiese tanto dibujar como ver lo que se había dibujado.

La manera de incorporar esta pizarra a tu aplicación y el buen diseño de sus clases hicieron que fuese muy sencillo de entender, utilizar y modificar en caso de que quisieras personalizarlo a tu gusto. Dispone de un cuadro de opciones donde poder elegir el tipo de objeto que queremos dibujar (mano alzada, rectas, círculos, cuadrados, estrellas... etc). En todas estas opciones puedes elegir además qué color quieres para el contorno o para el relleno, así como rehacer o deshacer en caso de haberte equivocado.

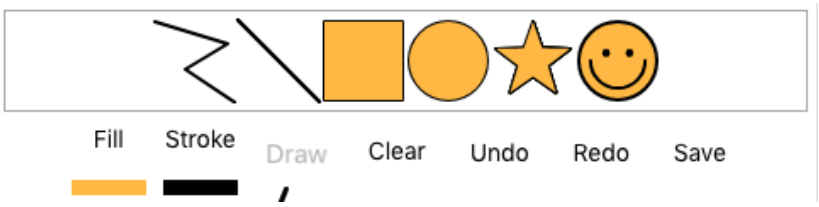


Figura 3.1: Opciones por defecto de Ng2 WhiteBoard

El número de objetos o formas que ofrece con esta librería es escaso, como vemos en la figura 3.1, podemos crear una línea, un dibujo a mano alzada, un círculo, un rectángulo, una estrella y una cara sonriente. Sin embargo, me faltaba algo que consideraba imprescindible para mi proyecto. No tenía la posibilidad de crear texto. Tras investigar, observé que todas las formas heredan de la misma clase por lo que me

fue sencillo crear una nueva forma que me permitiera insertar el texto que yo quisiera.

Al principio consideré que esta librería era una muy buena opción sobre la que desarrollar este proyecto.

El diseño y la arquitectura utilizada para implementar las funcionalidades básicas de la pizarra hicieron que el inicio de mi proyecto fuese fácil y sin complicaciones. La herencia entre las distintas clases de objetos y formas que poder usar en la pizarra simplificaba mucho el uso y desarrollo de esta librería. Sin embargo, esa misma cualidad de simplicidad que vi de forma positiva al inicio, se convirtió en algo negativo a la hora de querer modificar y ampliar las funcionalidades que ofrecía se volviese más costoso. Más adelante profundizaré sobre cuales fueron los inconvenientes encontrados y la solución propuesta.

3.4.2. Fabric.js

Fabric.js es una libreria de canvas en JavaScript y HTML5 que ofrece múltiples opciones y objetos para quien quiera incorporar una pizarra a su proyecto.

Disponen de una página web ([fab, 2023]) donde podemos encontrar una pizarra de demostración en la que probar las distintas funcionalidades que implementa esta librería. Además nos facilitan un apartado de tutorial donde te explican cómo utilizar cada componente de la pizarra y un apartado de documentación y de soporte por si tuviéramos dudas o problemas a la hora de implementarlo.

Esta librería se diferencia con la anterior en que los objetos que son dibujados en la pizarra se comportan como objetos independientes y una vez son dibujados podemos interactuar con ellos para cambiar sus propiedades como la posición, el tamaño o el color.

En su página web podemos ver el funcionamiento de la pizarra y su uso resulta muy sencillo y muy fácil de desarrollar.
La manera de incorporar la pizarra de Fabric.js a nuestra aplicación o sitio web es muy sencilla. Solo tenemos que declarar un lienzo en nuestro HTML utilizando la etiqueta canvas. En la figura 3-2 podemos ver un ejemplo de lo que se podría hacer en su página web con una pizarra desarrollada con esta librería.

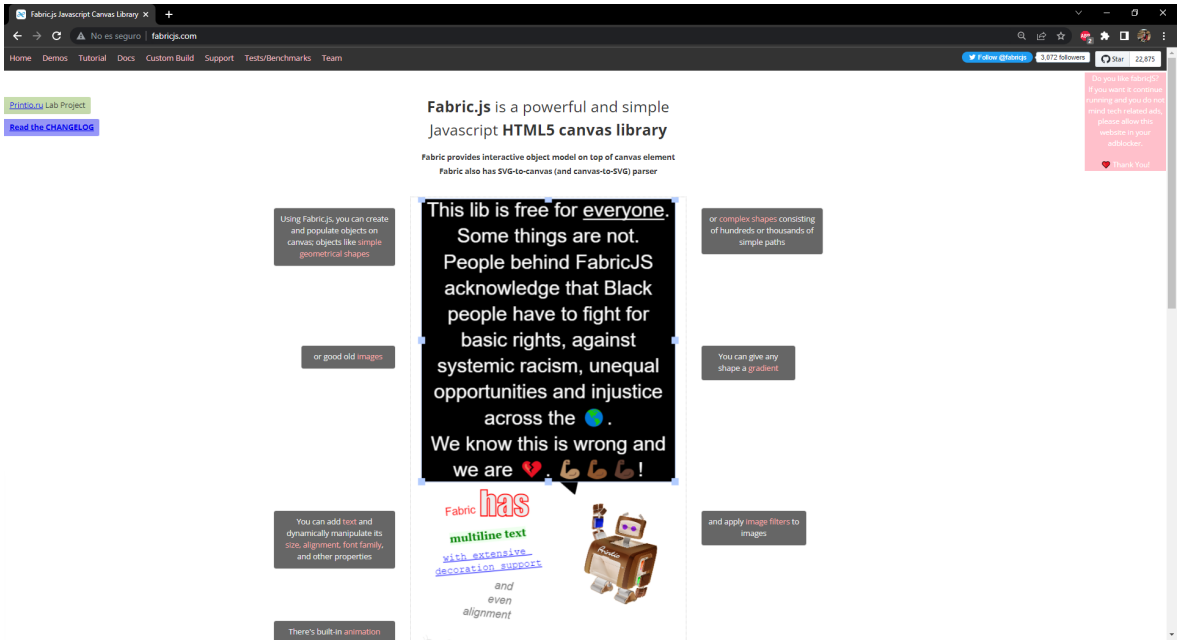


Figura 3.2: Página web de Fabric.js

A la hora de construir el componente Angular de la pizarra indicamos mediante código TypeScript que la etiqueta canvas que hemos declarado en nuestro el HTML es un objeto canvas de la librería Fabric.js, lo que hace que su uso sea muy snecillo que podamos así de fácil hacer uso de la librería.

3.4.3. Socket io

Tal y como describen de forma clara y concisa en su propio sitio web ([soc, 2023]), Socket.io es una librería que permite la comunicación de baja latencia, bidireccional y basada en eventos entre un cliente y un servidor

Trabaja con el protocolo WebSocket, utiliando una única conexión TCP enviando datos de forma bidireccional. En otras tecnologías habitualmente utilizadas, como Ajax

por ejemplo, el uso del protocolo HTTP obliga al cliente a pedir mediante peticiones datos al servidor y éste se los proporciona.

Con Socket.io, utilizamos un canal abierto bidireccional con protocolo TCP permitiendo así que el cliente no tenga que hacer peticiones cuando requiera algún recurso, sino que el servidor envía los datos requeridos en el momento que se necesite.

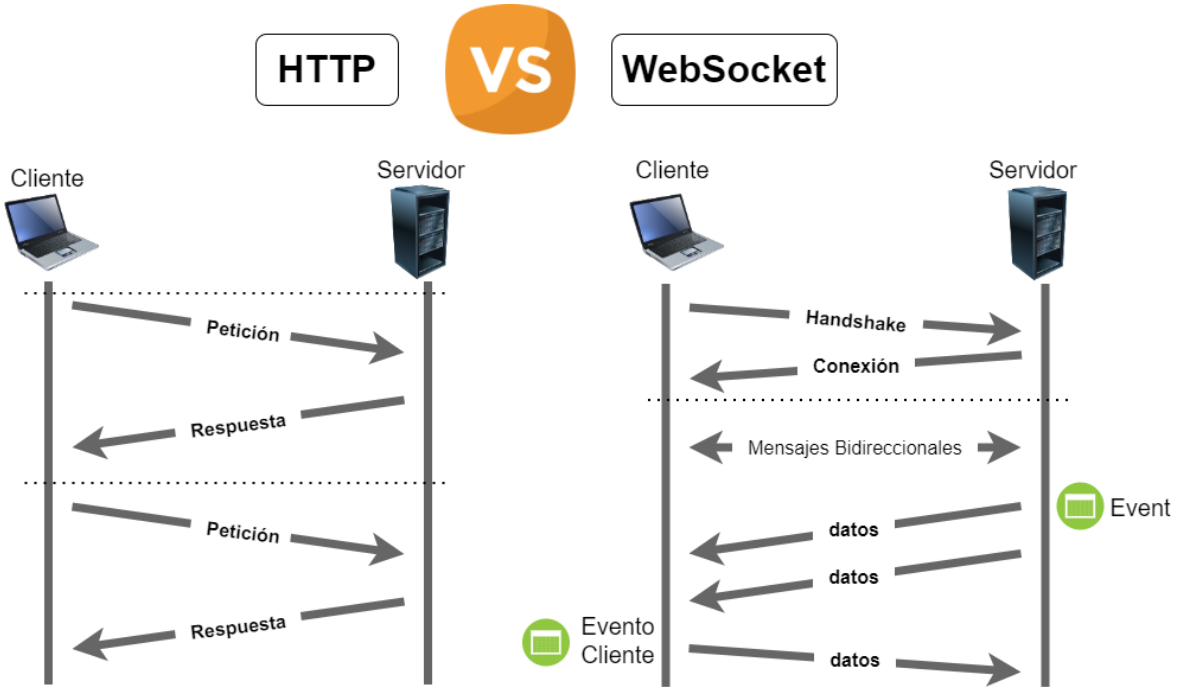


Figura 3.3: Http vs WebSocket

Tal y como podemos ver en la figura 3.3 mediante un sistema de eventos el servidor detecta que debe enviar datos y emite un evento que recibe el cliente. A su vez, el cliente puede también emitir y recibir eventos gracias a la comunicación bidireccional establecida.

3.4.4. Jest y Selenium

Jest y Selenium son las dos librerías que he utilizado para realizar los tests en mi proyecto.

Jest

Jest ([jes, 2023]) es una herramienta diseñada para ejecutar tests en JavaScript. Creado sobre Jasmine, otra librería de testing, y mantenido actualmente por Meta.
Fue desarrollado para satisfacer las necesidades específicas de Facebook en términos de pruebas unitarias y de integración en aplicaciones complejas basadas en JavaScript, como React.
Con Jest podemos ejecutar pruebas unitarias en paralelo de forma segura y probar nuestro código de manera que podamos desarrollar a medida que probamos nuestro proyecto. Se caracteriza entre otras cosas porque es compatible con Angular, React, NodeJS o VueJS y gracias a su facilidad de configuración y su velocidad de ejecución se ha convertido en una de las principales opciones entre los desarrolladores para ejecutar las pruebas.
Entre las principales ventajas que tiene Jest podemos encontrar:

1. Rapidez y eficiencia: Tiene una gran capacidad de ejecutar distintas funciones en paralelo, lo que lo hace muy útil para grandes proyectos con muchas funcionalidades.

2. Sintaxis sencilla: Ofrece una sintáxis clara y sencilla de forma que el desarrollador pueda entender cada una de sus funcionalidades

3. Funcionalidades integradas: Incluye diversas funcionalidades con la que poder verificar el resultado de nuestro código del proyecto, generar informes o objetos mocks para simular escenarios de prueba.

Por otro lado, como inconvenientes podemos destacar:

1. Configuración complicada: En algunas ocasiones puede resultar dificil llegar a configurar Jest correctamente en los proyectos. Debido a sus posibles personalizaciones y conceptos.

2. Curva de aprendizaje: Resulta algo complejo de entender en los inicios debido al primer inconveniente comentado de dificil configuración.

3. Entornos de prueba limitados: No permite la creación de pruebas para entornos como pruebas de backend o pruebas de aplicaciones móviles.

Selenium

Selenium ([sel, 2023]) es un entorno de pruebas para aplicaciones web que permite la automatización de pruebas unitarias y la simulación de interacciones de usuario en

diferentes escenarios de uso. Para poder intreractuar con la aplicación web he realizado pruebas unitarias automáticas con Selenium WebDriver abriendo distintos navegadores y navegar con ellos simulando la actividad de usuarios en la pizarra colaborativa, dibujando e interactuando con las distintas funcionalidades.

Sin embargo, una de las dificultades que me ha supuesto Selenium es la imposibilidad de acceder al objeto de Canvas de HTML ya que su contenido se dibuja utilizando gráficos y no elementos HTML regulares.

Para ello he tenido que hacer uso de la funcionalidad de ejecutar Javascript en el navegador y así poder acceder a las propiedades del canvas.

3.5. Metodología

Este proyecto lo he realizado siguiendo una metodología iterativa e incremental. Mi tutor y yo planificamos unos objetivos a lo largo de los meses, de manera que acordamos 3 entregas o releases del proyecto. Estas entregas las planteamos de la siguiente manera:

1. **Noviembre 2022** - Proyecto con Angular, comprobando que la librería Ng2-Whiteboard-Canvas es adecuada.
2. **Marzo 2022** - Proyecto con una demo funcional con funcionalidades básicas de pizarra digital funcionalidades extra.
3. **Mayo 2022** - Implementar posibles funcionalidades extra.
4. **Mayo 2023** - Aumentar test y funcionalidades exta.

En estas fechas la intención era ver el avance, alcance y los objetivos que se han podido cumplir y cuáles no para planificar la siguiente entrega.

El trabajo realizado con este trabajo ha sido iterativa e incremental, puesto que cada una o dos semanas, revisábamos el trabajo realizado y planificábamos nuevos objetivos.

Sin embargo, aun habiendo organizado las entregas así, se han visto modificadas a lo largo del proyecto debido a contratiempos que no me han permitido cumplir con las fechas señaladas. He hecho lo posible por sacar el trabajo adelante y poder realizar las entregas lo antes posible para dar finalizado el proyecto.

La metodología iterativa e incremental se ajustaba muy bien a las características de mi proyecto. Al no saber qué librería sería la más adecuada, he tenido que tomar decisiones a medida que lo iba realizando. Decisiones como por ejemplo qué funcionalidad era viable o inviable o qué librería podría ser más útil que otra.

En una metodología iterativa e incremental el cliente puede seguir el avance del proyecto ya que en cada entrega el proyecto tiene las funcionalidades de la entrega anterior más las propuestas de la actual. Además, en caso de requerir algún cambio, estos se pueden hacer a tiempo, a medida que el cliente, o en este caso entre mi tutor

y yo, viésemos oportuno añadir o quitar requisitos y funcionalidades.

Cabe destacar que la librería utilizada para el proyecto, Fabric.js, no era la primera opción por la que comencé su desarrollo.

Inicialmente investigué la posibilidad de crear una pizarra colaborativa a partir del proyecto que podemos encontrar en GitHub ([ng2, 2023]), el cual consiste en una pizarra desarrollada con Angular algo más simple y menos compleja que la que podemos encontrar con Fabric.js. Sin embargo, al ser más simple era más complicado modificarla y ampliarla para conseguir que fuese colaborativa con amplias funcionalidades. Es por ello que, en la segunda fase del proyecto, descarté esta idea y comencé a desarrollarlo con Fabric.js.

He utilizado Trello como herramienta para oganizar las diferentes tareas que iba programando a lo largo de las entregas. Con Trello he podido llevar un control sobre qué tareas tenía que realizar, cuáles estaba realizando en ese momento y cuáles estaban terminadas. De esta manera me he podido organizar y llevar un desarrollo ágil teniendo siempre presente el seguimiento de las tareas de mi proyecto.

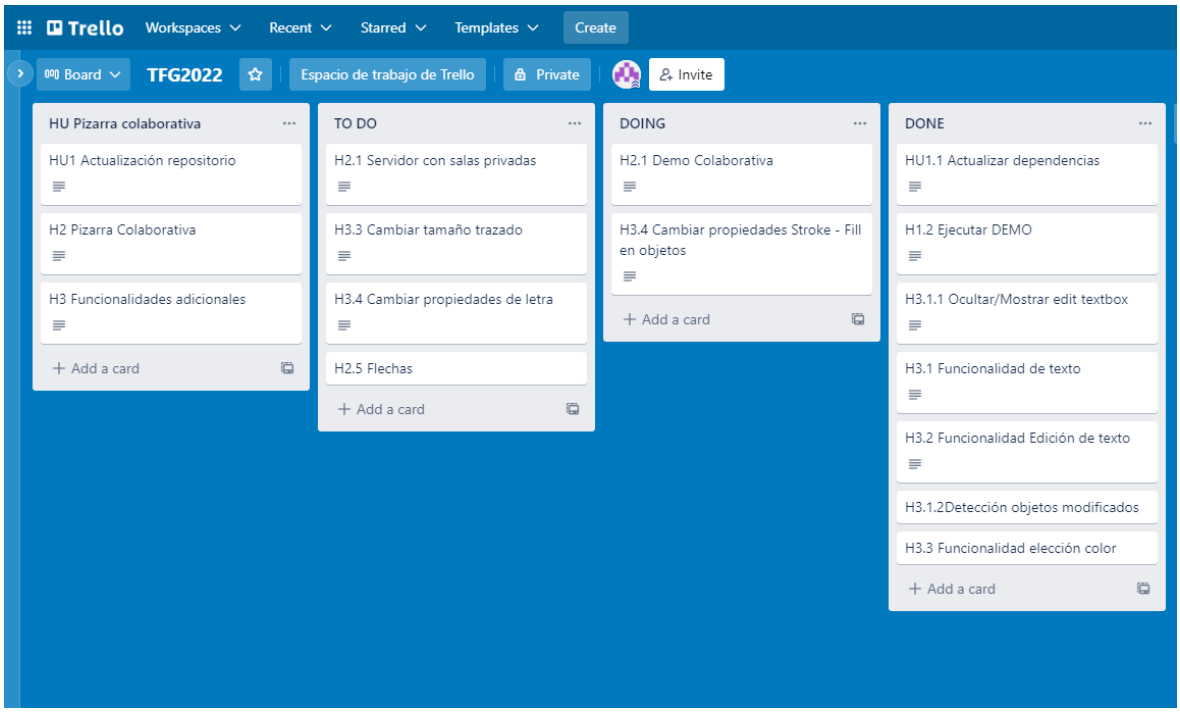


Figura 3.4: Trello Pizarra Colaborativa

Tal y como podemos ver en la figura 3.4, mi Trello consistía en 4 columnas. En la primera columna aparecen las Historias de Usuario que engloban los requisitos a partir

de los cuales genero las tareas.

En la segunda columna, creo las tareas a partir de las Historias de Usuario con una breve descripción detallada sobre el trabajo que tengo que realizar.

Las demás columnas son las que utilizo para saber el estado de las tareas, si estan realizándose o ya han sido terminadas.

De esta manera consigo tener un control y seguimiento de mi trabajo.

Capítulo 4
Descripción informática

Este trabajo ha consistido en realizar una pizarra colaborativa, desarrollada con Angular, con la que dos o más usuarios puedan interactuar a tiempo real. La conexión entre los usuarios se realiza gracias a NodeJs y Socket.io.

4.1. Requisitos

A continuación voy a exponer en forma de tabla los principales requisitos funcionales y no funcionales que antes de comenzar el proyecto planteamos para el desarrollo de la pizarra colaborativa en Angular.

4.1.1. Requisitos Funcionales

Los requisitos funcionales son las características o funcionalidades explicitas que cumple un software. Los requerimientos que desea el cliente en el desarrollo de un proyecto software.
Se plantean en el inicio del proyecto entre el cliente y el equipo del proyecto y plantan las bases de lo que queremos conseguir.

Codigo Requisito	RF01
Nombre	Salas online
Descripción	El usuario debe estar dentro de una sala para poder acceder a la pizarra colaborativa. Sólo se podrá utilizar la pizarra cuando el usuario esté conectado a una misma sala.
Prioridad	Alta

Codigo Requisito	RF02
Nombre	Dibujo
Descripción	La pizarra permitirá dibujar al menos los elementos simples de una pizarra digital convencional.
Prioridad	Alta

Codigo Requisito	RF03
Nombre	Borrado
Descripción	La pizarra permitirá borrar las figuras que se han dibujado en el lienzo.
Prioridad	Alta

Codigo Requisito	RF04
Nombre	Modificado
Descripción	La pizarra permitirá modificar la posición y el tamaño de las figuras que se hayan dibujado en el lienzo.
Prioridad	Alta

Codigo Requisito	RF05
Nombre	Dibujo a mano alzada
Descripción	La pizarra ofrecerá la funcionalidad de dibujo a mano alzada para hacer trazas libres con el movimiento del ratón.
Prioridad	Alta

Codigo Requisito	RF06
Nombre	Figura rectangular
Descripción	La pizarra ofrecerá la posibilidad de dibujar figuras rectangulares o de forma cuadrada.
Prioridad	Alta

Codigo Requisito	RF07
Nombre	Figura Circular
Descripción	La pizarra ofrecerá la posibilidad de dibujar figuras circulares.
Prioridad	Alta

Codigo Requisito	RF08
Nombre	Lineas rectas
Descripción	La pizarra ofrecerá la posibilidad de dibujar lineas rectas en cualquier direección.
Prioridad	Alta

Codigo Requisito	RF09
Nombre	Texto
Descripción	La pizarra ofrecerá la posibilidad de introducir texto.
Prioridad	Alta

Codigo Requisito	RF10
Nombre	Acciones Colaborativas
Descripción	Las acciones de dibujar, modificar o eliminar cualuqier figura en el lienzo por un usuario deben ser replicadas en las pizarras de los demás usuarios que estén conectados a la misma sala.
Prioridad	Alta

Codigo Requisito	RF11
Nombre	Editar texto
Descripción	El texto ya insertado en la pizarra podrá editarse para cambiar sus propiedades como tamaño tamaño color o tipo de fuente.
Prioridad	Media

Codigo Requisito	RF12
Nombre	Flechas
Descripción	La pizarra permitirá insertar flechas como las que podemos encontrar en diagramas o esquemas.
Prioridad	Media

Codigo Requisito	RF13
Nombre	Imágenes
Descripción	La pizarra permitirá insertar imágenes en el lienzo.
Prioridad	Baja

4.1.2. Requisitos No Funcionales

Los requisitos no funcionales se refieren a las características que debe poseer el proyecto en cuanto restricciones o cualidades, las cuales se plantean en el diseño de la arquitectura. Cómo se va a realizar el proyecto, qué tecnologías se van a usar o qué restricciones va a tener.

Codigo Requisito	RNF01
Nombre	Angular
Descripción	La pizarra consistirá en un componente web desarrollado con Angular que pueda ser integrada en otras aplicaciones
Prioridad	Alta

Codigo Requisito	RNF02
Nombre	Reutilización de código
Descripción	El proyecto se desarrollará con técnicas como el uso de herencia, técnicas SOLID u otros mecanismos que garanticen la reutilización de código con el objetivo de economizar tiempo, esfuerzo y redundancia.
Prioridad	Alta

Codigo Requisito	RNF03
Nombre	Escalabilidad
Descripción	El proyecto tenderá la capacidad de poder ampliar sus funcionalidades sin perder calidad. Ampliar en tamaño y funcionalidades de manera que se adapte a las necesidades del proyecto al que se incorpore esta pizarra.
Prioridad	Alta

Codigo Requisito	RNF04
Nombre	Robustez
Descripción	Se desarrollará un software robusto que no se vea comprometido en situaciones de fallas, ataques externos o uso erróneo de la aplicación con control sobre el servidor, el acceso de los distintos usuarios a las salas y la información que se transmiten entre ellos.
Prioridad	Alta

Codigo Requisito	RNF05
Nombre	Integridad
Descripción	Se garantizará la integridad de la información compartida entre los distintos usuarios, de manera que no afecte a la usabilidad. En caso de que se produzca un fallo de conexión o una excepción inesperada en el momento de dibujar, borrar o modificar un objeto de la pizarra la acción realizada debe ser corregida si no se ha podido hacer llegar a los demás usuarios.
Prioridad	Alta

Codigo Requisito	RNF06
Nombre	Usabilidad
Descripción	La pizarra será fácil de usar y el diseño será lo más intuitivo posible. El usuario será capaz de utilizar todas las funcionalidades de la pizarra sin ningún problema.
Prioridad	Alta

4.2. Arquitectura y Análisis

En este capitulo explicaré de forma detallada la arquitectura y el diseño del proyecto y las clases y los componentes que lo forman apoyándome en distintos diagramas y figuras que ayuden a entender el funcionamiento de esta pizarra colaborativa.

El funcionamiento es muy sencillo, ya que ocurre principalmente gracias a dos librerías, Fabric.js para usar la pizarra, y Socket.io para comunicar los cambios entre los distintos usuarios.

Con esta pizarra colaborativa el usuario es capaz de dibujar, borrar o modificar cualquier figura para que sea replicada en las pizarras de los demás usuarios que estén conectadas a una misma sala del servidor.

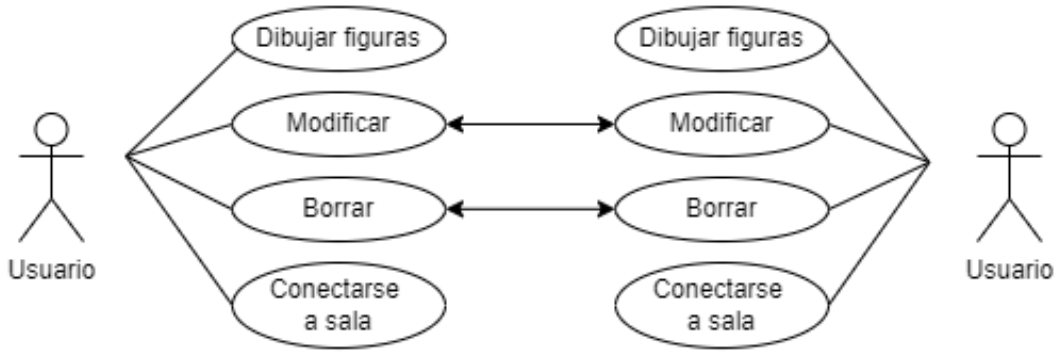


Figura 4.1: Caso uso funciones del usuario

Lo que quiero mostrar en la figura 4.1 del diagrama de caso de uso, es que el usuario no solo puede interactuar con las figuras que él mismo ha dibujado, sino que además existe la posibilidad de que otro usuario que esté en la misma sala modifique nuestra figura cambiando el color, la posición o el contenido en caso de ser un texto insertado. Esto permite más interactividad ya que en un mismo dibujo varios usuarios pueden intervenir, sin embargo, puede ser contraproducente si estamos en una situación en la que no deseamos que nadie modifique nuestra parte.

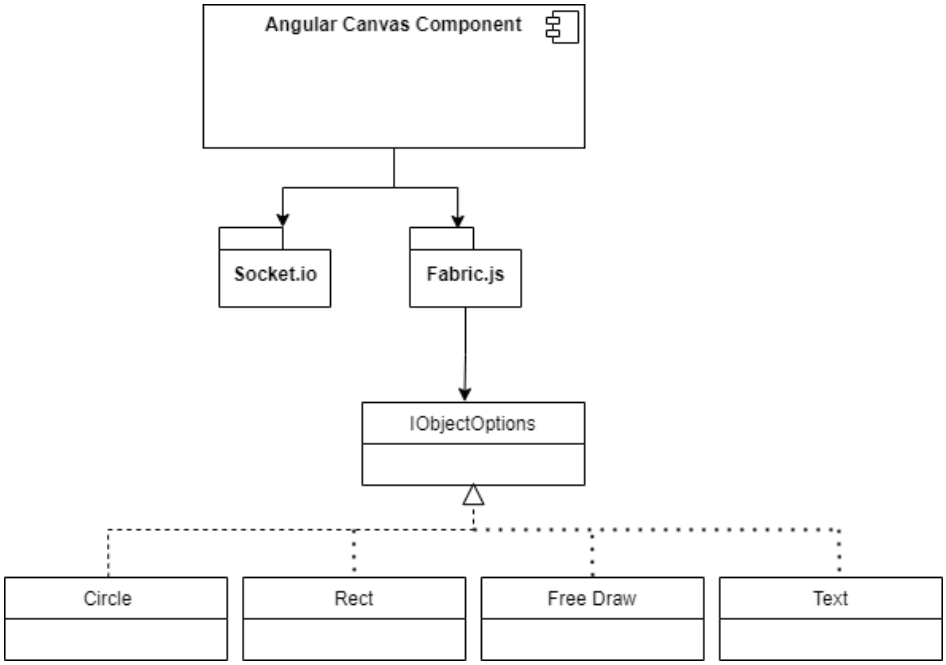


Figura 4.2: Diagrama de clases Socket y Fabric.js

Tal y como podemos ver en la figura 4.2, la pizarra usa principalmente la librería Fabric.js para utilizar sus objetos como círculo, rectángulo o texto entre otros. Estos implementan todos la misma interfaz IOjectOptions de manera que comparten las funcionalidades básicas y hace que el código sea mucho más eficiente, legible y reutilizable.

Para poder comunicar los cambios que se producen en la pizarra de un usuario, ya sea añadiendo, modificando o borrando figuras, uso la librería Socket.io que me permite comunicar esos cambios a los demás usuarios que estén conectados a la misma sala.

Cuando un usuario dibuja una figura en el canvas, gracias a los eventos que incorpora Fabric.js, el componente Canvas es capaz de detectar que ha ocurrido algo y podemos tratarlo. En función de qué evento se haya lanzado podremos avisar a los demás usuarios indicándoles qué cambio se ha realizado para que se replique en sus respectivas pizarras.

4.2.1. Análisis de clases

A continuación explicaré cada una de las clases y componentes que componen el proyecto de forma que se pueda entender claramente el funcionamiento. Desde las clases que componen el proyecto hasta las utilizadas para la comunicación entre los usuarios conectados.

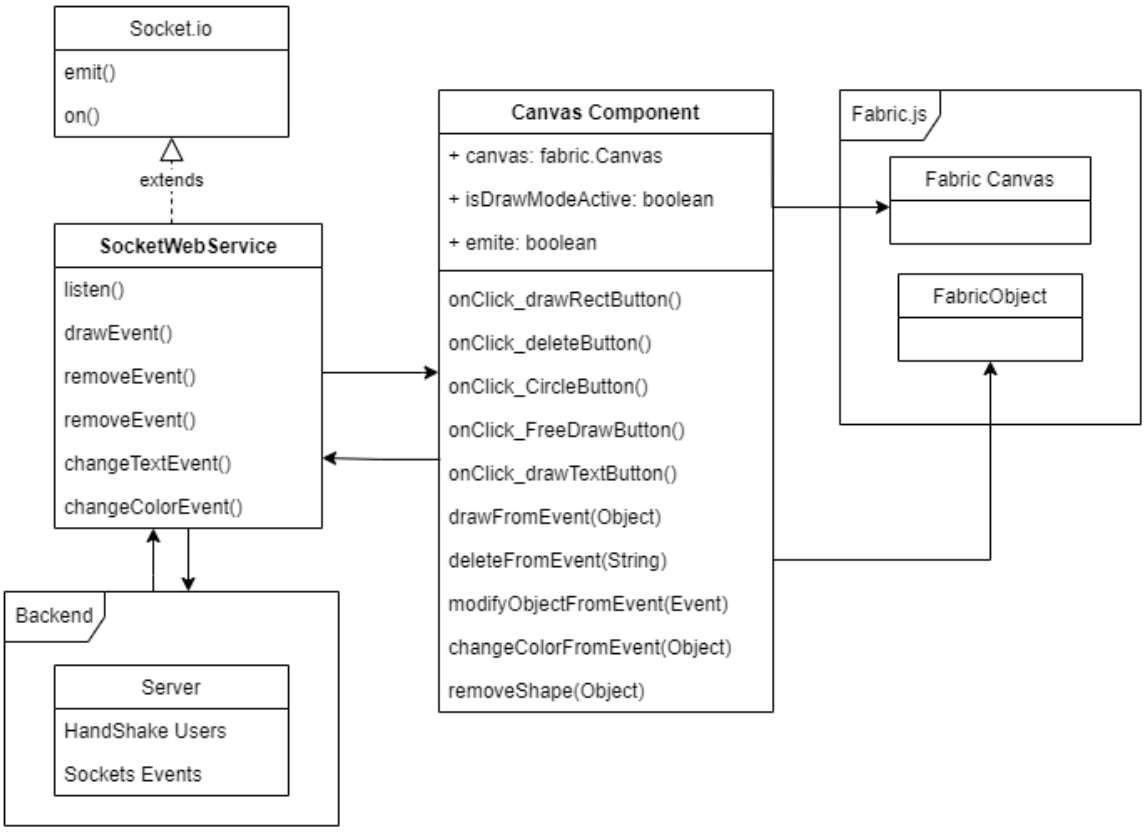


Figura 4.3: Diagrama clases del proyecto

Como podemos observar en la figura 4.3 la estructura de las clases de la pizarra colaborativa no es muy compleja. Consta de una clase CanvasComponent que hace uso de la libreria Fabric.js, y una clase SocketWebService que utiliza la libreria Socket.io para emitir y recibir eventos con los cambios realizados en las pizarras de los usuarios.

CanvasComponent

Es el componente Angular de la pizarra colaborativa. En él es donde he creado la pizarra propia de la librería Fabric.js y donde implemento todas las funcionalidades del proyecto que Fabric.js me ofrece.

Este componente utiliza la clase SocketWebService suscribiéndose a los eventos que pueda recibir y emitir con los posibles cambios. Tal y como podemos ver en el diagrama, he distinguido entre los distintos métodos que puedan ejecutarse para poder diferenciar si el cambio realizado en la pizarra ha sido del usuario en su propia pizarra, o de lo contrario es un cambio producido en la pizarra de otro usuario que ha de replicarse en

la suya.

SocketWebService

La clase SocketWebService es la clase que se encarga de emitir y recibir los eventos producidos en las pizarras de los usuasrios conectados.

Hereda de la clase Socket que nos encontramos en la librería Socket.io. Con ella podemos hacer uso de eventos que nos facilitan la comunicación entre los usuarios. Cuando un usuario ha hecho un cambio en la pizarra le comunica a SocketWebService que se ha realizado un cambio. Por ejemplo, se dibujado un círculo y se llama al método drawEvent() del SocketWebService.

Este servicio se encarga de emitir un evento al servidor con el que, gracias a Socket.io y la tecnología de WebSocket, transmitimos la información del cambio realizado a las demás pizarras que se encuentren conectadas a la misma sala. Lo mismo ocurrirá en el caso de borrar una figura, donde se llamará al método removeEvent(), cambiar la posición o escribir, cada funcionalidad está asociada a su propio evento.

Backend Server.js

Esta clase está escrita en JavaScript. Se encarga de gestionar la parte del servidor de este proyecto. Hace uso de la librería Socket.io creando un servidor al que han de conectarse los usuarios para poder utilizar la pizarra colaborativa.

En este fichero JavaScript creamos un servidor que nos permite comunicarnos entre los diferentes clientes y el servidor de forma bidireccional emitiendo y recibiendo eventos.

4.3. Diseño e implementación

En este apartado explicaré de forma más detallada la implementación y el diseño de la pizarra colaborativa con Angular. Durante el proceso de este proyecto no he ido siempre en la misma dirección, ni ha consistido simplemente en coger una pizarra y hacerla colaborativa . Como todo proyecto software, el desarrollo ha tenido una fase de

investigación, planteamiento del alcance del proyecto, toma de decisiones de posibles alternativas, desarrollo y pruebas.

Durante el inicio del proyecto, la idea era utilizar una librería que implementaba una pizarra bastante simple y básica, lo cual consideraba buena idea ya que como punto inicial sobre el que partir siempre es atractivo algo sencillo.

Sin embargo a medida que desarrollaba el proyecto y podíamos ampliar los requisitos y funcionalidades, vi que la pizarra base utilizada se nos empezaba a quedar un poco corta para llegar a nuestro objetivo. Por ello, decidimos mi tutor y yo contemplar la posibilidad de utilizar otra librería con más funcionalidades y mucho más escalable aprovechando lo que ya había aprendido con la primera versión.

A continuación explicaré de forma detalla la primera versión, explicando sus ventajas e inconvenientes.

4.3.1. Primera versión: NG2 Canvas Whiteboard

La pizarra que nos podemos encontrar en este proyecto [ng2, 2023] es una pizarra básica y sencilla sobre la que comenzó todo. Se trata de una pizarra que dispone de las funcionalidades más básicas de una pizarra digital, dibujo a mano alzada y algunas figuras.

La manera de incorporar esta pizarra a una aplicación Angular es sencilla. Basta con añadir a tu código HTML su componente 'canvas-whiteboard', el cual cuenta gracias a Angular con distintas propiedades para elegir las opciones y que quieres añadir o quitar al lienzo.

```
<div style="width: 300px; height: 300px; border: 1px solid black">
    <canvas-whiteboard #canvasWhiteboard
        [options]="canvasOptions"
        [drawButtonClass]="'drawButtonClass'"
        [drawButtonText]="'Draw'"
        [clearButtonClass]="'clearButtonClass'"
        [clearButtonText]="'Clear'"
        [undoButtonText]="'Undo'"
        [undoButtonEnabled]="true"
        [redoButtonText]="'Redo'"
        [redoButtonEnabled]="true"
        [fillColorPickerText]="'Filled'"
        [strokeColorPickerText]="'Stroked'"
        [colorPickerEnabled]="true"
        [lineWidth]="5"
        [strokeColor]="'rgb(0,0,0)'"
        [shouldDownloadDrawing]="true">
    </canvas-whiteboard>
</div>
```

Código 4.1: Declaración de un canvas con sus propiedades

Además, esto también es personalizable desde el propio constructor del componente del canvas, CanvasWhiteboardComponent, por si quieres añadir, quitar o modificar las modalidades de forma dinámica en tu aplicación.

```
constructor(private canvasWhiteboardService: CanvasWhiteboardSl
    constructorervice, private canvasWhiteboardShapeService:
    CanvasWhiteboardShapeService){

this.canvasWhiteboardShapeService.unregisterShapes([CircleShape,
    SmileyShape, StarShape, LineShape, RectangleShape]);
 }
```

Código 4.2: Constructor de Canvas personalizable

Posee distintas figuras para dibujar en el canvas como son el cuadrado, circulo, línea, cara sonriente o mano alzada. Todas estas formas están implementadas de manera que heredan de la misma clase CanvasWhiteBoardShape. Esto permite que el código se entienda mejor, sea más usable y lo más importante, permite reutilización y escalabilidad.

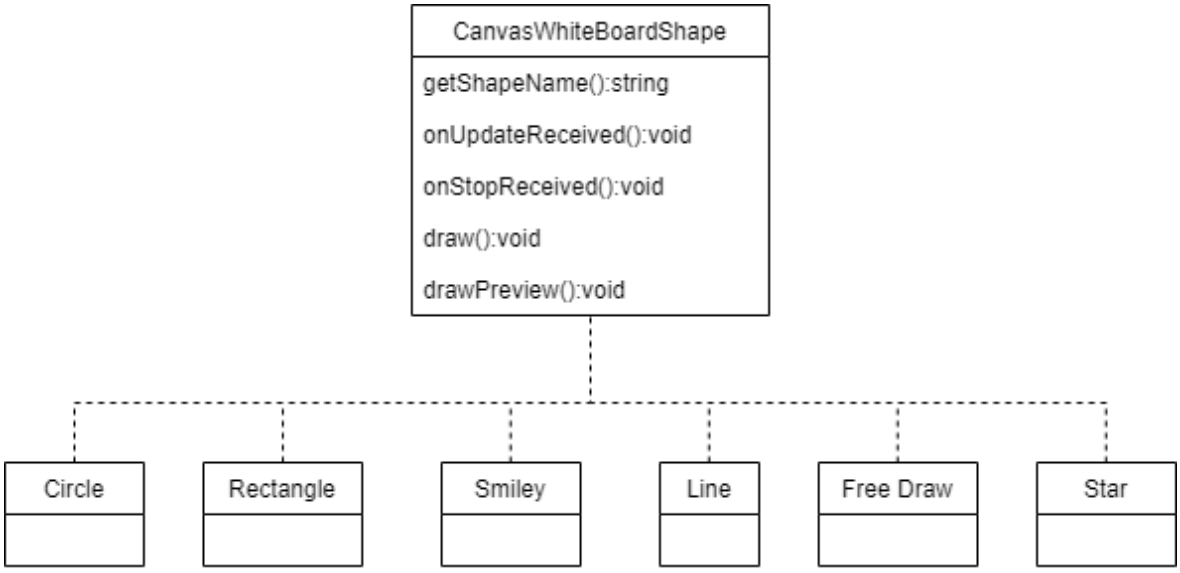


Figura 4.4: Diagrama clases de objetos Fabric

Gracias a que las figuras están implementadas así, donde cada forma es una clase que hereda de la misma clase Shape (ver figura 4.4) fue sencillo incorporar una funcionalidad nueva que carecía desde el principio y considerábamos un requisito importante, el cuadro de texto. Esta funcionalidad la desarrollé creando una nueva forma que heredase CanvasWhiteBoardShape de la misma forma que lo hacen las demás.

Sin embargo la forma de utilizar esta herramienta no era muy cómoda ya que se comportaba como si fuese una figura en vez de un verdadero cuadro de texto. Si quería escribir algo concreto implementé un método que enlazara lo que el usuario quisiera escribir en un TextBox de la aplicación antes de dibujar el texto en la pizarra.

Vi entonces que esta librería me limitaría muchos de los requisitos y funcionalidades que queríamos desarrollar para este proyecto, por lo que después de reunirme con mi tutor acordamos utilizar otra librería aparentemente más completa y compleja.

4.3.2. Segunda versión: Fabric.js

Habiendo descartado la librería anterior para este proyecto, planteamos de nuevo el desarrollo para la siguiente entrega. El objetivo era aprovechar lo ya aprendido con la nueva versión para poder aplicarlo con la librería Fabric.js y conseguir tener una pizarra colaborativa con Angular.

Con la versión anterior, conseguí implementar una pizarra digital básica, un servidor al que poder conectar distintos usuarios, y que las funcionalidades existentes se pudiesen

replicar entre las distintas pizarras conectadas entre sí.

La librería Fabric.js posee múltiples funcionalidades que podemos implementar tal y como podemos ver en www.fabricjs.com. Después de investigar unos días acerca de esta librería y de las posibilidades que ofrece, descubrí que incorpora distintos eventos que detectan los cambios que se producen en la pizarra.

4.3.3. Eventos Fabric.js y Socket.io

Fabric.js incorpora una gran lista de eventos que nos permiten detectar en cualquier momento los cambios que se han producido en la pizarra.
Entre los más útiles o los principales podemos destacar los eventos objeto añadido, modificado y borrado. `('on(object:added",e)` `('on(object:modified",e)`, `('on(object:removed",e)`

Estos eventos nos ayudan a saber cuándo el lienzo ha sido modificado y, con la ayuda de Socket.io, comunicamos los cambios a los demás usuarios.

Con la librería Socket.io he implementado una serie de métodos que emiten un evento al servidor con la información del cambio producido. El servidor al recibir el evento, emite a todos los usuarios conectados en la sala el objeto y la acción que se ha producido.

Me encontré con el problema de no poder referenciar a las figuras entre los usuarios porque no poseían ninguna propiedad identificativa.

Cuando un usuario modificaba un objeto presente en su lienzo, podía transmitir mediante el evento el objeto modificado con sus nuevos valores, pero las pizarras de los demás usuarios no sabían a qué objeto aplicar esos cambios.

Por ello implementé un método que asociase un código identificador Uuid único a cada una de las figuras que el usuario dibuja. De esta manera, cuando me comunico con el servidor para enviarle el objeto que se ha modificado envío también el identificativo que tiene. Así los demás usuarios utilizan este identificativo para encontrar la figura deseada en su canvas.

```
1 generateObjectId(): string | undefined {
2     return uuid.v4();
3   }
4
5 getObjectById(n: string): fabric.Object {
6     return this.canvas.getObjects().find((object: any) => object.name
          === n);
7   }
```

Código 4.3: Función para generar y obtener a los objetos por su identificador

Tal y como podemos ver en este ejemplo con el método `generateObjectId()` se genera un identificativo Uuid que se asocia al objeto nuevo. Con el método `getObjectById()` buscamos entre ellos el identificativo que corresponda con la figura que deseamos obtener.

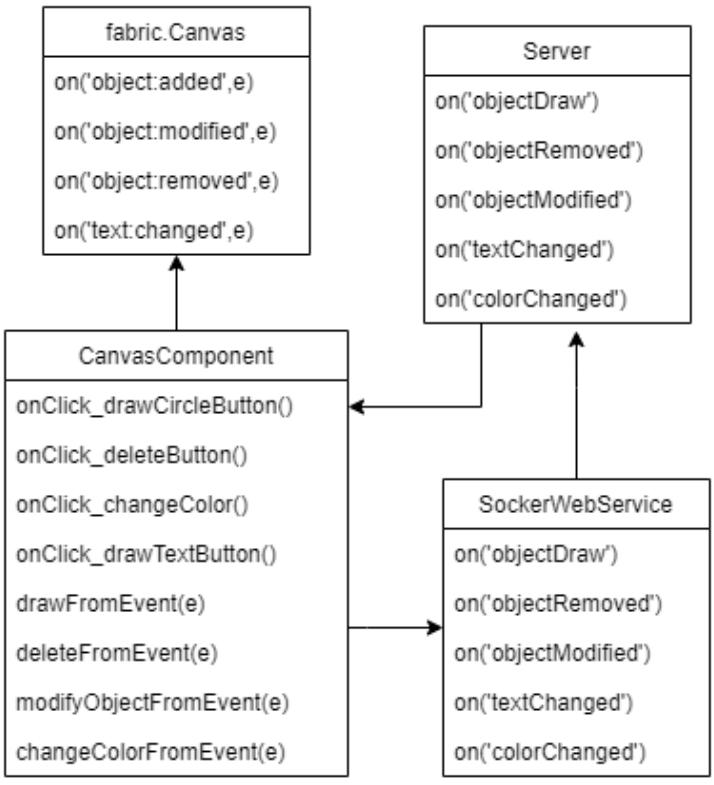


Figura 4.5: Diagrama de clases

En este diagrama de la figura 4.5 podemos enteder gráficamente cómo el componente de la pizarra hace uso de los eventos de la librería Fabric.js para detectar los cambios. Éste llama al servicio SocketWebService para avisar del cambio producido y emite un evento al servidor para que lo comunica al resto de usuarios.

El sistema se identifica como una arquitectura cliente-Servidor, donde cada usuario conecta de forma bidireccional con el servidor gracias a la tecnología WebSocket y los eventos.

Este sistema se comporta como si fuese una arquitectura de eventos donde existe un emisor, un gestor que maneje los eventos producidos y suscriptores que reciban la información. Sin embargo, al ser una comunicación bidireccional todos los servidores son al mismo tiempo productores como suscriptores de eventos.

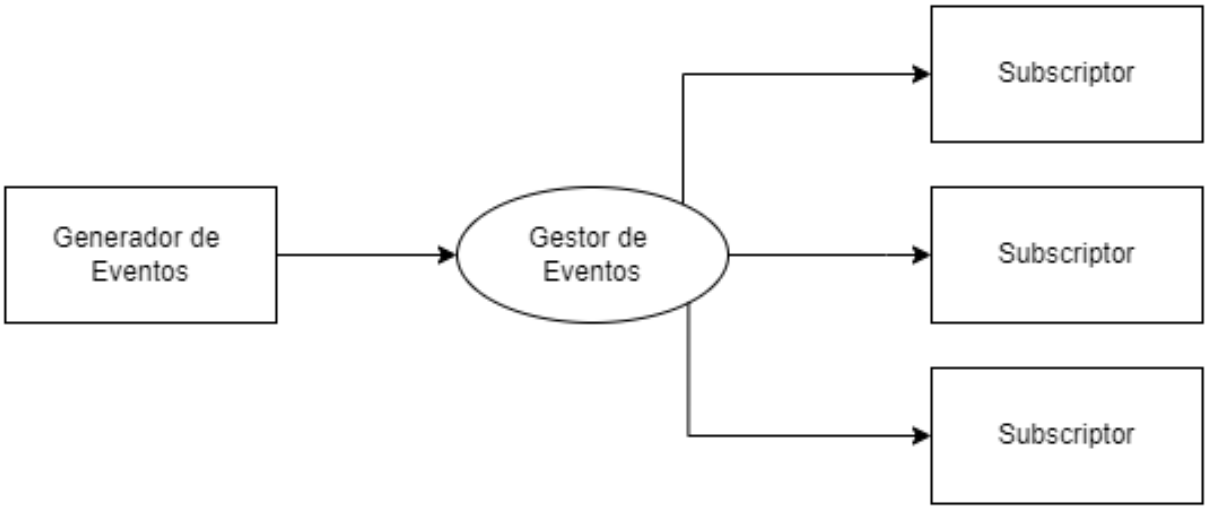


Figura 4.6: Diagrama arquitectura de eventos

Este diagrama de la figura 4.6 sería una explicación gráfica de cómo un usuario es productor de eventos que mediante un gestor, en mi caso SocketWebService, controla y maneja los eventos para transmitirlos a los demás usuarios.

En este punto, puede surgir una duda, ¿cómo puede saber el sistema quién ha realizado el evento para no comunicárselo y que se produzca el cambio dos veces en su pizarra?

Mediante una variable local que posee cada componente o usuario, a modo de 'flag' o 'bandera', controlo quién ha emitido el evento del cambio para que a él no le afecte y así evitar que se produzca el cambio dos veces.

Es decir, si el usuario A realiza un cambio en el canvas, su variable indicará que ha sido él el emisor poniéndose a 'true'. De manera que cuando el servidor emita a todos

los usuarios su cambio, él lo ignorará puesto que ha sido él el emisor.

De no controlar este caso, se provocaría un bucle infinito en el que un usuario emite un cambio, a la vez el servidor le avisa del cambio producido y él volvería a realizar el cambio en su pizarra.

Para explicar el funcionamiento de mi proyecto de forma gráfica haciendo uso de la librería Fabric.js y del servidor desarrollado con Socket.io, voy a apoyarme en el siguiente diagrama.

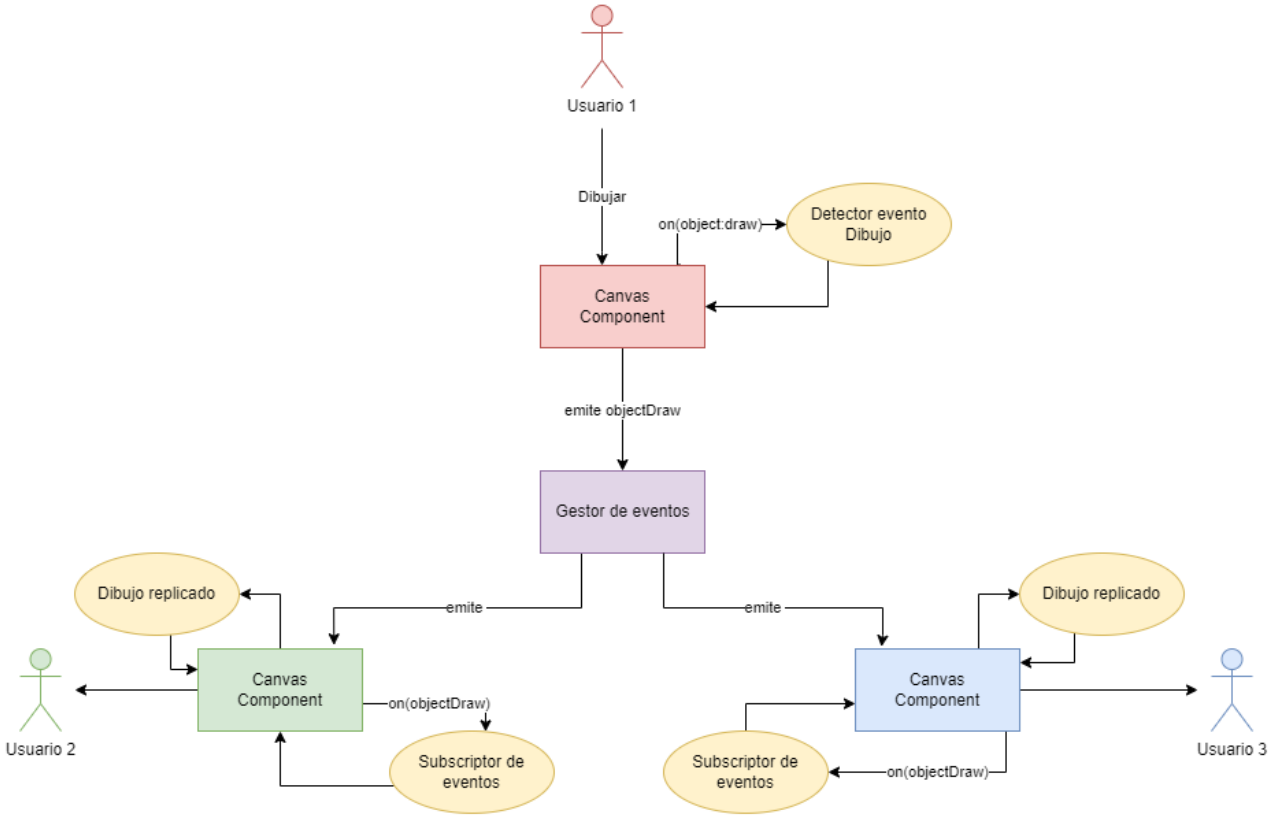


Figura 4.7: Diagrama demostrativo flujo de eventos

En esta otra figura 4.7, en el diagrama representado he intentado representar cúal sería el flujo de eventos desde que el usuario dibuja, modifica o borra algo en la pizarra hasta que es comunicado a los demás usuarios para que efectúen el cambio.

En este ejemplo podemos ver cómo el Usuario 1 realiza un cambio en el canvas. Dibuja un objeto nuevo.

Su propio canvas, gracias a la librería Fabric.js, detecta la nueva figura dibujada gracias a su evento on(object:added) y mediante código hacemos saber a nuestro

servicios de eventos el cambio realizado.

Nuestro gestor de eventos, SocketWebService, se encarga de emitir el evento al servidor para comunicarle que se ha dibujado una figura nueva, y este emite el evento que recibirán todos los usuarios conectados, en este caso, Usuario 2 y Usuario 3. Al estar suscritos todos a los eventos emitidos por el servidor, reciben la información necesaria para saber qué acción se ha producido, qué objeto se ha visto afectado y poder replicarlo en sus pizarras.

De otra forma, para completar la explicación del flujo de eventos entre los distintos componentes de la aplicación, he diseñado dos diagramas de secuencia para ver de forma clara cuáles son los mensajes que se transmiten entre el usuario que interacciona con la pizarra y el servidor que comunica los cambios a los demás usuarios.

De esta manera, en la figura 4.8 podemos ver los mensajes enviados entre el usuario que emite un nuevo cambio y el servidor.

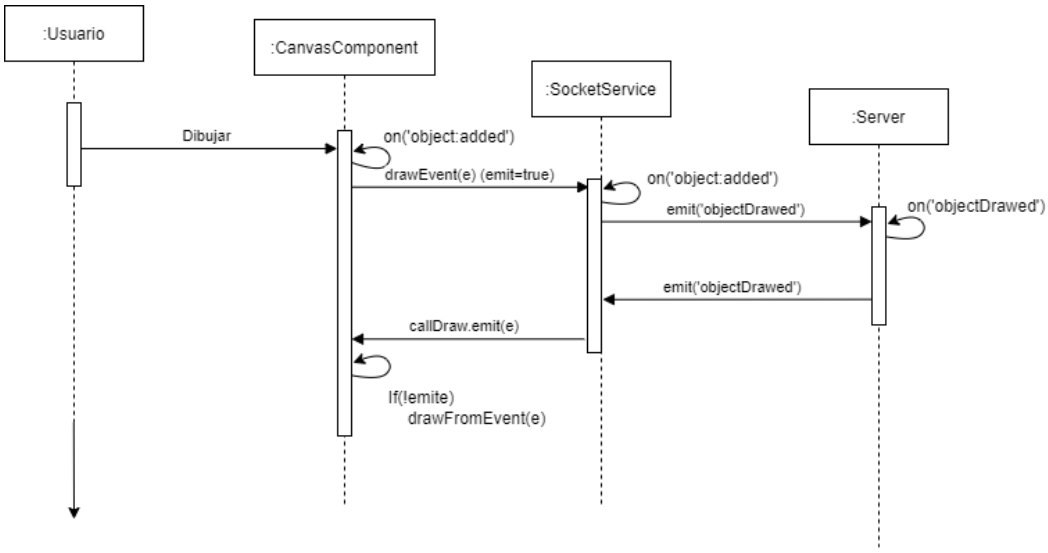


Figura 4.8: Diagrama de secuenca del emisor

y en la figura 4.9 los mensajes enviados del servidor a los demás usuarios que no son emisores del cambio.

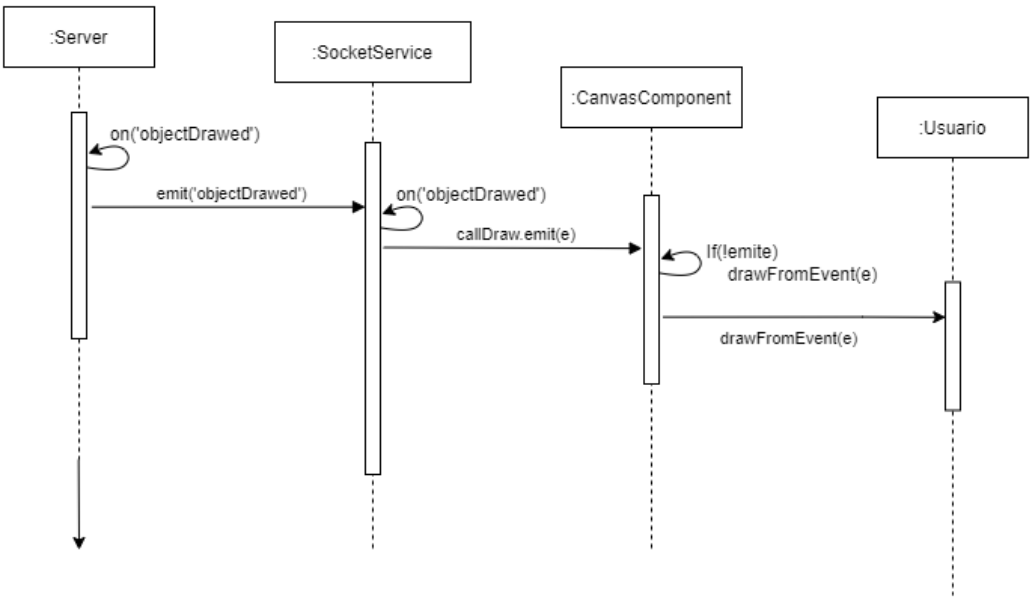


Figura 4.9: Diagrama de secuenca del receptor

Tal y como podemos ver en la figura 4.10, la aplicación Demo que he implementado para comprobar el correcto funcionamiento de la pizarra posee los botones correspondientes a todas las funcionalidades. Tenemos un botón para borrar, dibujar a mano alzada, insertar un cuadro de texto e incluso cambiar el color del pincel. Además tenemos un botón para elegir una figura donde elegiremos entre un rectángulo, un circulo y una línea.

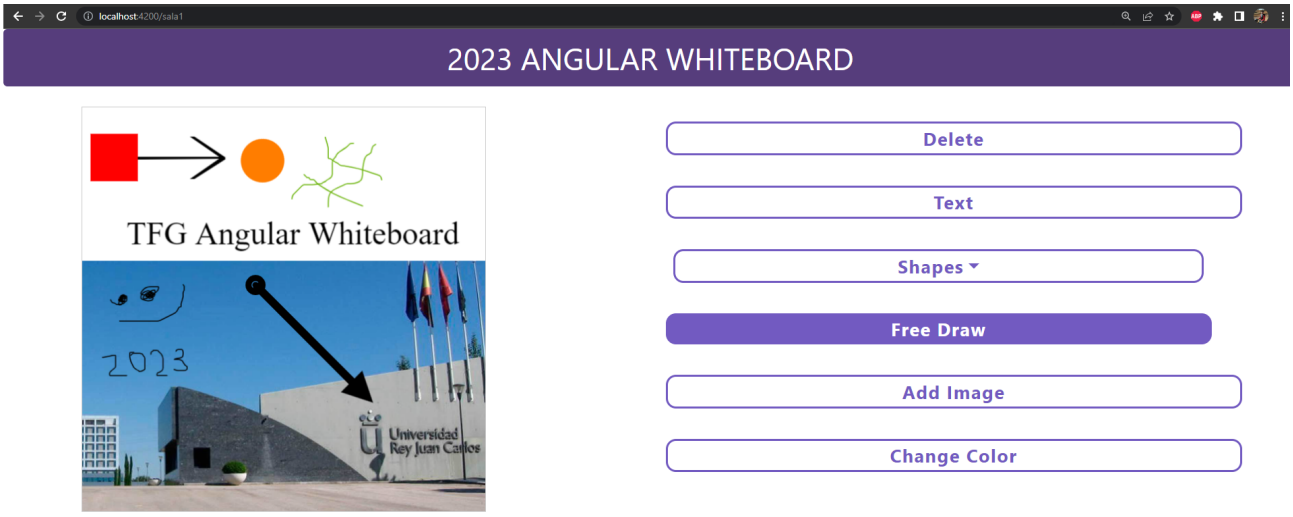


Figura 4.10: Aplicación de Demo

4.4. Pruebas

En este capítulo voy a explicar qué he utilizado para realizar las pruebas de la pizarra y cómo funciona el testing.

Por un lado he utilizado Jest, una herramienta que nos permite probar código JavaScript. He configurado el proyecto de manera que no referencie la librería Jasmine que viene por defecto en los proyectos Angular para que pueda dirigirse con Jest.

```
1  "scripts": {
2    "ng": "ng",
3    "start": "ng serve",
4    "build": "ng build",
5    "watch": "ng build --watch --configuration development",
6    "test": "jest"
7  },
8  "jest": {
9    "testPathDirs": ["src"],
10   "setupTestFrameworkScriptFile": "src/app/test.js"
11 },
```

Código 4.4: Configuración entorno test Jest

Para realizar las pruebas, he utilizado la librería Selenium Testing que me permite automatizar funcionalidades de aplicaciones web.

Con Selenium podemos probar nuestro código de aplicaciones web abriendo navegadores y creando test que comprueben el correcto funcionamiento de nuestra aplicación.

En el caso de este proyecto he realizado un test en el que abrimos dos navegadores Chrome y probamos que las funcionalidades se ejecutan sin problemas tanto en el usuario que dibuja como en el que recibe.

Para ello he creado un test en el que haciendo uso de la tecnología de Selenium WebDriver abrimos dos navegadores Chrome, les redirigimos a una sala en concreta y tratándolos de forma independiente, un driver dibujará fromas en su canvas y otro driver dibujará lo mismo en el suyo.

```
1 let driver1 = new Builder().forBrowser('chrome').build();
2 await driver1.get('http://localhost:4200/sala1');
3
4 let driver2 = new Builder().forBrowser('chrome').build();
5 await driver2.get('http://localhost:4200/sala1');
6 });
```

Código 4.5: Abrir dos navegadores con Selenium Webdriver

Con las siguientes lineas, podemos ver cómo indicar a cada driver qué tiene que dibujar, comparando primero que todos los pixeles de ambos canvas sean iguales ya que no se ha dibujado nada aún.

```
1  await driver1.wait(until.elementLocated(By.id('canvas')), 60000);
2  await driver2.wait(until.elementLocated(By.id('canvas')), 60000);
3
4  var canvas1Data = await driver1.executeScript('return document.
       getElementById("canvas").getContext("2d").getImageData
       (0,0,400,400);').then(async e => {
5      return e
6  });
7  var canvas2Data = await driver2.executeScript('return document.
       getElementById("canvas").getContext("2d").getImageData
       (100,100,2,2);').then(async e => {
8      return e
9  });
10 let areEqualsBefore = JSON.stringify(canvas1Data.data) === JSON.
       stringify(canvas2Data.data);
11 if (!areEqualsBefore){
12     driver1.close();
13     driver2.close();
14     throw new Error("Canvas at these points are not equeals!");
15 }
```

Código 4.6: Comparar Canvas vacíos y pintar un cuadrado en Driver 1

Con Selenium Webdriver no podemos acceder al codigo en HTML del canvas ya que el los elementos dibujados en un Canvas se generan dinámicamente utilizando Javascript. Es por ello, que usamos el método 'executeScript' para poder acceder a las entrañas de nuestro canvas.

Una vez teniendo nuestra pizarra, podemos recuperar el valor de sus píxeles para asegurarnos que son todos 0, es decir, están todos los píxeles en blanco. La gracia de este test, es comparar el valor de los píxeles habiendo clickado en dibujar una única pizarra, viéndose reflejado el resultado en ambas gracias a los eventos.

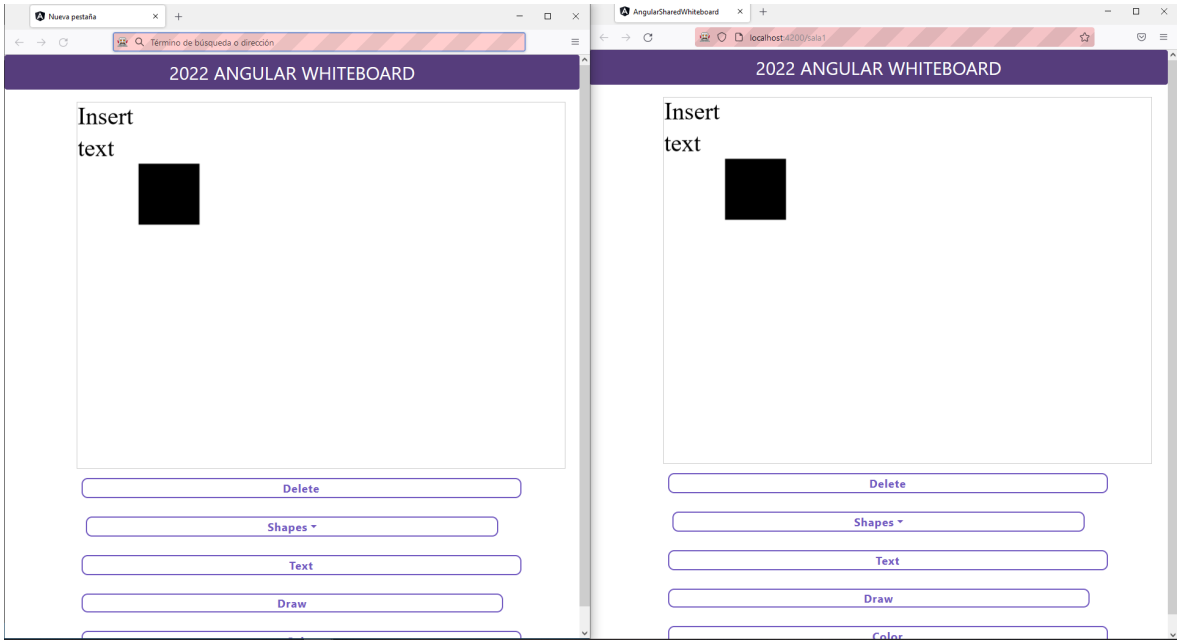


Figura 4.11: Ejemplo de 2 pizarras tras ejecutar el test

Tras haberse pintado la figura en un solo driver (una sola pizarra), volvemos a preguntar por el valor de los píxeles de la pizarra, comparándolos entre sí para ver que los píxeles de la pizarra originalmente dibujada son exactamente iguales que los píxeles de la que no se ha dibujado nada de forma directa.

```
 let btnShapes = await driver2.findElement(By.id('btnshapes'));
await btnShapes.click();

let btnShapeRect = await driver2.findElement(By.id('btnRect'));
var rectDrawed = await btnShapeRect.click().then(async clicked => {
   return clicked;
});
console.log(areEqualsBefore);
let areEqualsAfter = false;
var cv1data = await driver1.executeScript('return document.
   getElementById("canvas").getContext("2d").getImageData
   (0,0,400,400);').then(async e => {
   return e
});

console.log('cv1data drawed: ', cv1data.data);
var cv2data = await driver2.executeScript('return document.
   getElementById("canvas").getContext("2d").getImageData
   (0,0,400,400);').then(async e => {
   return e
});
console.log('cv2data drawed: ', cv2data.data);
areEqualsAfter = JSON.stringify(cv1data.data) === JSON.stringify(
   cv2data.data);
if (!areEqualsAfter) {
   driver1.close();
   driver2.close();
   console.log('cv2data drawed: ', cv1data.data);
   console.log('cv2data drawed: ', cv2data.data);
   throw new Error("Canvas at these points are not equeals!")
}


driver1.close();
driver2.close();
```

Código 4.7: Pintar en en el primer Driver y corroborar mediante los píxeles que se ha pintado en el segundo también

```
Test Suites: 6 passed, 6 total
Tests:       7 passed, 7 total
Snapshots:   0 total
Time:        41.494 s, estimated 49 s
Ran all test suites.
```

Figura 4.12: Test ejecutados con éxito

Como podemos ver en la figura 4.12 aparecen los test comlpetados con éxito. El proyecto ejecuta los tests propios de los diferentes componentes que componen la aplicación de Angular como el componente de la 'home' o la propia pizarra, y por último ejecuta el script ya mencionado desarrollado con Selenium.

Capítulo 5
Conclusiones y trabajos futuros

Al comienzo del proyecto supe que el camino sería largo y costoso, pero con trabajo y sacrificio sabía que podía finalizar el proyecto cumpliendo con las necesidades y los requisitos requeridos.

Considero que se ha cumplido el objetivo inicial de crear una pizarra colaborativa desarrollada en Angular, con la tecnología que nos ofrece Socket.io y después de haber probado diferentes opciones y librerias. Por otro lado, podría decir que se pueden implementar más funcionalidades u opciones que permitan a la pizarra utilizarse de manera más intuitiva y fácil.

El objetivo principal de crear una pizarra colaborativa era claro desde el principio, sin embargo el alcance del proyecto, las funcionalidades finales y en general la forma de realizarlo no siempre estuvo claro.
Ha sido un proceso largo, con problemas y soluciones, con toma de decisiones para poder continuar en la dirección correcta y sin duda ha sido un desafío tanto académico como personal.

El problema de no saber si la librería inicial con la que empecé sería la adecuada o si las funcionalidades requeridas en los requisitos serían posibles de implementar hizo que tuviera que realizar un trabajo previo de investigación. Dedicar tiempo a diseñar el proyecto a alto nivel fue algo clave en el desarrollo del proyecto.

A medida que iba desarrollando la pizarra y encontrándome con los primeros problemas hizo que más de una tarde me viera incapaz de avanzar yo solo, pero con trabajo y confianza pude ir resolviendo todos los inconvenientes.

Me ha ayudado a aumentar mis conocmientos sobre Angular y la tecnología WebSocket. Entender correctamente el funcionamiento de los eventos en las aplicaciones web, cómo se transmiten información entre el cliente y el servidor.
Además he aprendido nuevas herramientas para el testing de código software, especialmente para aplicaciones web como es Jest y Selenium, puesto que no lo había utilizado anteriormente.
En definitiva este proyecto me ha hecho crecer como persona y como desarrollador y me siento muy orgulloso de lo que he conseguido y aprendido durante estos años en la carrera, tanto por la gente, compañeros y profesores, como la materia aprendida y superada.

5.1. Limitaciones

Cabe destacar, que como trabajos futuros o limitaciones del proyecto, la falta de alguna funcionalidad que se pueda considerar básica en una pizarra, como poder editar la fuente de un texto o los bordes de figuras.
Se podrían implementar en un futuro más figuras que nos podemos encontrar en la librería Fabric.js. Tanto figuras como herramientas que permiten que la pizarra sea más útil e interactiva pero que por problemas de tiempo no he podido implementar. Entre las posibles limitaciones existen, puedo señalar la falta de posibilidad de editar las características como el borde de las figuras o el tipo de letra al escribir.

También puedo destacar como problemas el hecho de haber podido resolver el modificado de varios objetos al mismo tiempo.
No he sido capaz de averiguar la forma en la que hacer saber a los usuarios conectados qué objetos se han modificado cuando se trata de varios objetos a la vez. Si un usuario mueve un único objeto, las nuevas propiedades de ese objeto las puedo replicar en las demás pizarras. Sin embargo, cuando el usuario selecciona con el ratón más de una figura, las propiedades de la posición del lienzo se ven alteradas por la propia selección del ratón, por lo que no he sido capaz de transmitir a los demás usuarios las coordenadas finales de los objetos al ser movidos.

Bibliografía

[js, 2023] (2023). Info Javascript. https://developer.mozilla.org/es/docs/Web/JavaScript.

[ng2, 2023] (2023). Repositorio Whiteboard NG2. https://github.com/webfactorymk/ng2-canvas-whiteboard.

[ang, 2023] (2023). Sitio web de AngularJs. https://www.angular.io.

[fab, 2023] (2023). Sitio web de Fabricjs. https://www.fabricjs.com.

[git, 2023] (2023). Sitio web de Github . https://www.github.com.

[nod, 2023] (2023). Sitio web de NodeJs . https://www.node.org.

[soc, 2023] (2023). Sitio web de Socket IO. https://www.socket.io.

[ts, 2023] (2023). Sitio web de Typescript. https://www.typescriptlang.org/.

[jes, 2023] (2023). Sitio web Jest. https://jestjs.io/es-ES/.

[sel, 2023] (2023). Sitio web Selenium. https://www.selenium.dev/.

[Scozzafava, 2020] Scozzafava, A. (2020). Presentación de la encuesta sobre teletrabajo en españa — cámara de españa. *Camara de Comercio*.