Universitat Oberta de Catalunya

uoc.edu

Proyecto de desarrollo web: VirtualClass

Manuel Sánchez Cañadas
Grado en Ingeniería Informática (2do semestre, curso 2019-2010)
Desarrollo web

Responsable de la asignatura: Gregori Robles Martínez

12 de junio de 2020

FICHA DEL TRABAJO FINAL

Título del trabajo:	*VirtualClass*
Nombre del autor:	*Manuel Sánchez Cañadas*
Nombre del consultor/a:	*Gregori Robles Martínez*
Nombre del PRA:	*Gregori Robles Martínez*
Fecha de entrega (mm/aaaa):	*06/2020*
Titulación o programa:	*Grado en Ingeniería Informática*
Área del Trabajo Final:	*Desarrollo web*
Idioma del trabajo:	*Castellano*
Palabras clave	*Videollamada, Videoclase, Canvas.*

Resumen del Trabajo (máximo 250 palabras):

La realización de este proyecto de final de grado, pretende por un lado plasmar los conocimiento adquiridos en los últimos años, así como tratar de explorar y añadir nuevos conocimientos. Para ello se ha optado por el desarrollo de una plataforma web.

La creación de esta aplicación web surge con la necesidad de poner en contacto profesores particulares y alumnos sin necesidad de desplazarse.

El objetivo principal de este proyecto es la creación de una plataforma web clases virtuales, en la que los usuarios (profesor y alumno) estén conectados por videoconferencia. Así como proporcionar una herramienta de dibujo durante la clase, que puede ser utilizada en el caso de ser necesarias explicaciones complementarias de manera visual.

A modo de complementar la plataforma se ha añadido también una opción de Chat, que permite la comunicación entre profesor y alumno mediante un sistema de mensajería instantánea. Además, otras funcionalidades (como la de Calendario o Documentos) no se han podido añadir finalmente a la plataforma final por diversos problemas surgidos a lo largo del desarrollo y no disponer del tiempo suficiente para solucionarlos.

Este es un resumen básico de la aplicación web; sin embargo, se debe tener en cuenta que existen una serie de requisitos para cada funcionalidad y ha sido necesario el uso de diferentes tecnologías, que quedan más ampliamente detallados a lo largo de este documento.

Abstract (in English, 250 words or less):

This end-of-degree project aims to capture the knowledge acquired in recent years, as well as trying to explore and add new knowledge. To do this, a web platform has been developed.

The creation of this web application arises from the need to put private teachers and students in contact without having to travel.

The main objective of this project is the creation of a web platform for virtual classes, in which users (teacher and student) are connected by videoconference. As well as providing a drawing tool during the class, which can be used in case complementary explanations are needed in a visual way.

As a complement to the platform, a Chat option has also been added, which allows communication between teacher and student through an instant messaging system. In addition, other functionalities (such as the Calendar or Documents) have not been finally added to the final platform due to various problems that have arisen throughout the development and not having enough time to solve them.

This is a basic summary of the web application; however, it must be taken into account that there are a series of requirements for each functionality and it has been necessary to use different technologies, which are more extensively detailed throughout this document.

Índice

Lista de Figuras

Lista de Tablas

1. Introducción

1.1 Contexto y justificación del trabajo

VirtualClass es una aplicación web de clases virtuales, en la que los usuarios están conectados por videoconferencia. La idea principal es crear una plataforma web que permita a profesores particulares conectarse mediante videoconferencia con sus alumnos, así como proporcionar una herramienta de dibujo en el caso de ser necesarias explicaciones complementarias de manera visual.

1.2 Objetivos del Trabajo

Aplicación web con dos tipos de roles: profesor y alumno.

- El ***profesor*** puede crear sus clases, indicando las materias de las cuales puede ofrecer formación.
- El ***alumno*** estará suscrito a un profesor o varios de ellos.

En la plataforma existirá un chat entre el profesor y el alumno para comunicarse.

Una vez el profesor inicia la clase invita al alumno a participar en ella mediante video-conferencia.

La clase dispondrá de los siguientes servicios:

- Videoconferencia
- Chat
- Canvas para dibujar

1.3 Enfoque y método seguido

La metodología utilizada para la realización del presente trabaja se fundamenta en desarrollar un producto nuevo, habiendo sido para ellos necesario la investigación y recopilación de material bibliográfico enfocado a las tecnologías necesarias para el desarrollo de este proyecto.

1.4 Planificación del trabajo

1.4.1. Tecnologías que se van a utilizar

Servidor: Linux [1]

Servidor http: Nginx [2]

Base de datos: MongoDb [3] y/o Redis [4] o una similar para implementar una tabla hash si escalamos los servicios, saber en qué nodo está el usuario conectado por websocket, en el objetivo inicial no estará presente el escalado

Control de versiones: Git [5]

Despliegue: Docker [6] con (Docker compose o kubernetes si escalamos)

Backend:

- Nodejs [7]
- Express [8]
- Socket.io [9] o websocket nativo o alguna librería similar
- Mongoose [10] o driver nativo de mongodb en nodejs

Frontend:

- Html [11]
- CSS [11, 12]
- Javascript [11,13]
- Vue [14]
- Vuex [14]
- Vue-router [14]
- Axios [15]
- Otras librerías js

Comunicación:

- Rest [16]
- Websocket [17]
- RabbitMQ [18]

VideoConferencia: WebRTC [19,20]

1.4.2. Planificación temporal

Sprints (10 días aproximadamente cada uno de los siguientes sprints):

- Planificar cada servicio o componente del backend (user, class, chat, wsservice ...): casos de uso, modelo de datos, modelado de las apis (rest, websocket, rabbitmq), etc.
- Programar librerías o microframework común a cada servicio, preparar contenedores docker, preparar una *api gateway* para llamar a los servicios y preparar la arquitectura del servidor.
- Programar los servicios iniciales (user, class) con sus respectivos *test*.
- Iniciar el desarrollo del front con vue, con sus llamadas a los servicios básicos
- Programar servicio de chat y el de websocket. Desarrollar las vistas frontend del nuevo servicio.
- Programar el servicio de clase con la videollamada (webrtc) y sincronizar con el servicio de websocket.
- Programar servicios que falten (canvas) o extras (calendario, documentos).

1.4.3. Evaluación de riesgos

	Probabilidad	Nivel de riesgo	Impacto	Reducir el riesgo
Planificación no ajustada a la realidad	Media	Medio	Rediseñar la planificación y ajustarla a lo realizado.	Dedicar más horas a la planificación inicial.
Retraso en algún sprint	Media	Medio	Dilatación en los tiempos.	Derivar tareas a *sprints* con menor impacto.
Tecnología no se adapte a la necesidad	Bajo	Alto	Desarrollo extra, consumo de recursos excesivo, buscar nuevas tecnologías	Estudiar si las tecnologías se adaptan a las necesidades o casos de uso antes de implementarlas.
Falta de conocimiento técnico de las tecnologías utilizadas	Media	Alto	Tiempo en estudio de las tecnologías y mal uso de ellas.	Formar al equipo en las tecnologías que se van a utilizar.
Modelado erróneo del modelo de datos o APIs	Media	Alto	Rehacer modelos de datos y APIs.	Realizar modelos UML y estudiar impacto.

Tabla 1. Evaluación de riesgos.

1.5 Breve resumen de los productos obtenidos

Se ha desarrollado una plataforma que permite realizar clases online a nivel particular entre profesor y alumno a través de una videollamada. La plataforma también dispone de una herramienta de dibujo (Canvas) en el caso de ser necesarias explicaciones complementarias, así como una chat y una herramienta de almacenaje de documentos.

1.6 Breve descripción de otros capítulos de la memoria

Desarrollada la introducción y planteados los objetivos, a continuación, en los siguientes capítulos se recopilan:

- Requisitos del proyecto: en este apartado se especifican los diferentes requisitos y su alcance de cada una de las funcionalidades de la plataforma.
- Diseño del proyecto: se detallan los bocetos de diseño; tanto del logo, como de la web principal y la plataforma.
- Desarrollo del proyecto: se explica el proceso de desarrollo de la plataforma.
- Fase de pruebas: se detallan los *tests* realizados y sus resultados.

2. Requisitos del proyecto

Antes de empezar a implementar la aplicación, se establecieron una serie de requisitos para especificar su alcance. En este apartado se definen las funcionalidades mínimas que tendrá la aplicación.

2.1. Lista de requisitos

La plataforma web tiene las siguientes funcionalidades:

1. Crear cuenta de usuario

Req 1.1. Registro del usuario, formulario de alta de usuario (alumno o profesor)

Req 1.2. Formulario de alta de usuario tiene los siguientes campos: nombre de usuario que deberá ser único, email de usuario y contraseña cifrada por seguridad. El el caso del registro del perfil de profesor existirá un menú con una serie de asignaturas, que el usuario deberá seleccionar aquellas de las cuales tiene capacidad para impartir clases sobre dicha materia.

Req 1.3. Se redirige al formulario de acceso.

2. Conexión a la aplicación

Req 2.1. El formulario de acceso tiene dos campos de texto: nombre de usuario y contraseña. Además, tiene un botón de enlace de registro.

Req 2.2. Al identificarse, se redirige al perfil del usuario

3. Perfil de usuario

Req 3.1. El menú de usuario contiene las siguientes páginas: Videollamada (que únicamente se puede iniciar con el perfil profesor), Chat, Calendario (opcional), Documentos (opcional), Buscar profesor/alumno, Configuración, Cerrar sesión.

Req 3.2. Al hacer clic en “cerrar sesión”, el usuario se desconecta y vuelve a la página de inicio.

4. Buscar profesor/alumno

Req 4.1. Listado de alumnos o profesores disponibles.

Req 4.2. Al pinchar en un alumno o profesor, se ve la información básica del alumno o profesor y un botón para solicitar “alumno” o “profesor”

Req 4.3. Al pinchar en solicitar “alumno” o “profesor” se le envía una solicitud al alumno o profesor por correo y se publica en solicitudes pendientes.

Req 4.3. Si el usuario tiene solicitudes pendientes se le muestra un listado de los alumnos o profesores que se lo han solicitado, con un botón de aceptar o rechazar.

Req 4.3. Al aceptar la solicitud el usuario es vinculado al que ha realizado la solicitud (profesor o alumno), para poder chatear o hacer videollamadas.

Req 4.3. Al rechazar la solicitud se elimina de la lista de solicitudes pendientes.

5. Videollamada

5.1. Iniciar videollamada

Req 5.1.1. (profesor) Listado de alumnos conectados

Req 5.1.2. (profesor) Al hacer *click* sobre el perfil de un alumno, se redirige a la vista de la videollamada. Se le envía una notificación (*websocket*) al alumno para iniciar la videollamada.

Req 5.1.3. (alumno) Recibe una notificación en pantalla para iniciar o rechazar la videollamada.

5.2. Videollamada

Req 5.2.1. Comunicación con el servidor cada “x” segundos para verificar que todavia estoy en videollamada, si se pierde la comunicación con el servidor se cierra la videollamada.

Req 5.2.2. El navegador solicita permisos de uso de cámara y micrófono.

Req 5.2.3. Al aceptar permisos de cámara y micrófono, se envían los datos constantemente por *WebRTC* al alumno o profesor conectado con una conexión p2p. En el caso que esté conectado, si no cuando se éste conecte.

Req 5.2.4. Si el usuario rechaza los permisos de uso de cámara y micrófono se le notifica que no puede establecer la comunicación por falta de permisos.

5.3. Colgar videollamada

Req 5.3.1. En la vista de la videollamada hay un botón para colgar la videollamada, cuando se pulsa se cierra la videollamada y se redirige a la vista principal de la plataforma.

5.4. Chat en videollamada

Req 5.4.1. Listado de mensajes entre el alumno y el profesor. Debajo del listado hay un formulario con un botón enviar, al introducir texto en el formulario y pulsar enviar, se enviará el texto al otro usuario.

Req 5.4.2. Al recibir el texto del mensaje (*websocket*) se publica en la lista de mensajes con el avatar y nombre del usuario, así como la fecha y hora a la que ha sido enviado dicho mensaje.

5.5. Canvas en videollamada

Req 5.5.1. Canvas que al hacer *click* o arrastrar dibuja una línea, la línea dibujada se envia al otro usuario de la videollamada.

Req 5.5.2. Al recibir la línea dibujada del otro usuario (*websocket*) se publica en el canvas.

5. Chat

Req 5.1. Listado de alumnos o profesores con los que puedes chatear

Req 5.2. Al pinchar en un alumno o profesor se redirige a la vista del chat

Req 5.1. En la vista del chat, aparece un listado con los últimos mensajes enviados, ordenados de más reciente a más antiguo. Debajo del listado hay un formulario con un botón enviar.

Req 5.1. Al introducir texto en el formulario y pinchar en enviar, se envía el texto y se registra. Si el profesor o alumno está conectado lo recibe por *websocket*.

Req 5.1. Si el alumno o profesor está conectado y recibe un mensaje por *websocket*, si se encuentra en la vista del chat se publica en lista de mensajes, si no le aparece una notificación.

6. Calendario (opcional)

Esta funcionalidad se desarrollará y se determinarán los requisitos de ésta si tras el desarrollo de la función videollamada (y canvas) se dispusiese de tiempo para ello.

7. Documentos (opcional)

Al igual que la función "Calendario", la funcionalidad "Documentos" se desarrollará y se determinarán los requisitos de ésta si tras el desarrollo de la función videollamada (y canvas) se dispusiese de tiempo para ello.

3. Diseño del proyecto

Planificar un proyecto web puede ser una tarea compleja, y para ello es imprescindible saber definir las distintas tareas que deben realizarse en cada fase del proyecto web. Respecto al diseño del proyecto VirtualClass se han dividido tres subtareas: diseños del logo, diseño de la web principal y diseño y estructuración de la plataforma.

3.1. Diseño del logo

Un logo es un símbolo que representa tanto a tu marca como a la personalidad de ésta a través de una imagen lo más simple posible.

Figura 1. Logo del proyecto.

Para este diseño se ha elegido utilizar simplemente el nombre de la marca. Se ha elegido aplicar un color diferente a cada una de las palabras: gris para

Virtual y verde oscuro para *Class*; así mismo, se destaca la palabra Class con negrita ya que de este modo también se destaca la funcionalidad principal de la plataforma.

3.2. Diseño de la web principal

El diseño web abarcan diferentes aspectos como el diseño gráfico web, diseño de interfaz y experiencia de usuario, como la navegabilidad, interactividad, usabilidad, arquitectura de la información.

En este caso se ha optado por un diseño sencillo, que en un solo pantalla sin necesidad de hacer *roll* se puede ver todo el contenido de la web principal. Se han elegido tonos de colores claros ya que son más agradables para la vista y facilitan la lectura de los textos.

Figura 2. Diseño de la web principal.

Así mismo, esta página principal tendrá un diseño web adaptable (*responsive*), que tiene como objetivo adaptar una misma página web a distintos tamaños de pantalla, dispositivos y orientaciones.

3.3. Diseño y estructura de la plataforma

En diseño y estructuración de la plataforma web es una de las tareas más importantes en este proyecto. Dicha plataforma debe ser intuitiva y con una fácil usabilidad para el usuario. Para ello se ha optado por un menú a la izquierda de la pantalla donde se encuentran todas las opciones que el usuario usuario puede utilizar (videoclase, chat calendario, documento, ver o buscar profesores/alumnos), mientras que en el resto de la pantalla se encuentra una interfaz donde aparece contenido según aquella función que haya seleccionado el usuario en el menú de la izquierda

Figura 3. Diseño y estructura de la plataforma.

3.4. Diseño de la videoclase

Para la visualización de la plataforma de videollamada partimos del diseño de la plataforma (descrito en el apartado anterior), y en este caso en la interfaz de trabajo encontramos la videollamada y un chat. En caso de querer utilizar el Canvas para realizar alguna explicación adicional, se haría click en el botón del lápiz que se encuentra debajo de la videollamada, y la interfaz cambiaria por la del Canvas.

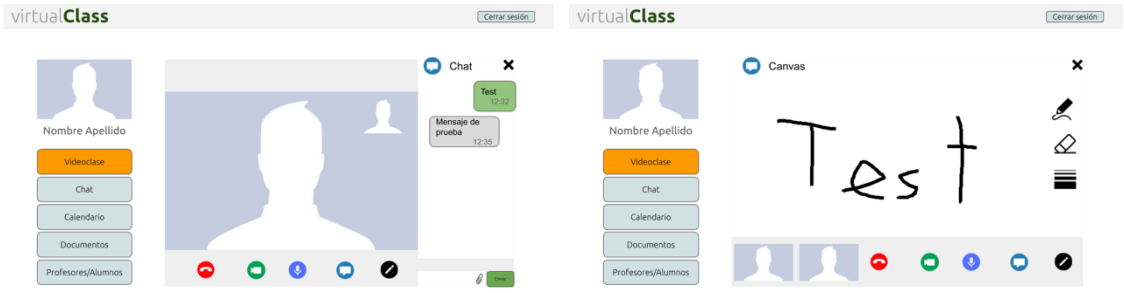


Figura 4 y 5. Diseño de la vista de la videoclase.

3.5. Diseño del chat

Del mismo modo que el anterior, en la interfaz de trabajo encontraríamos el Chat. En este caso al seleccionar en el menú la opción Chat aparecería en esta interfaz un listado con los diferentes usuarios con lo que se tiene relación. Para iniciar o continuar una conversación se debe seleccionar el usuario deseado, y de nuevo la interfaz cambia por un Chat con dicho usuario.

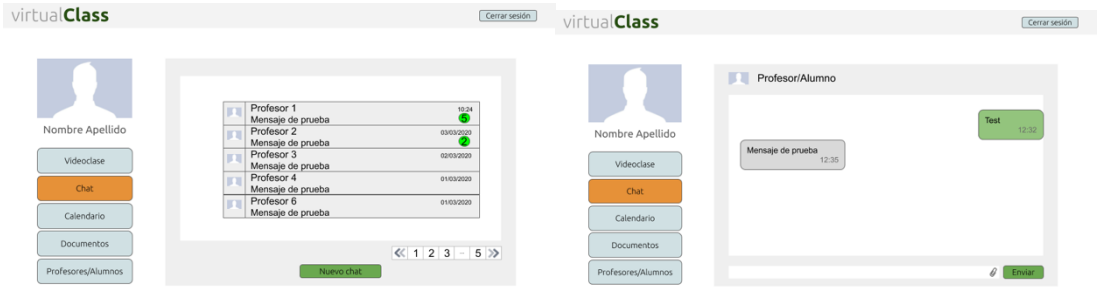


Figura 6 y 7. Diseño de la vista del chat.

3.6. Diseño del apartado Documentos

La opción de Documentos tiene un funcionamiento similar al del Chat, al seleccionar en el menú la opción Documentos aparecería en esta interfaz un listado de documentos que se han subido a la plataforma y que los usuarios pueden consultar. En este caso, al seleccionar un documento se descarga en el dispositivo que el usuario esté utilizando.

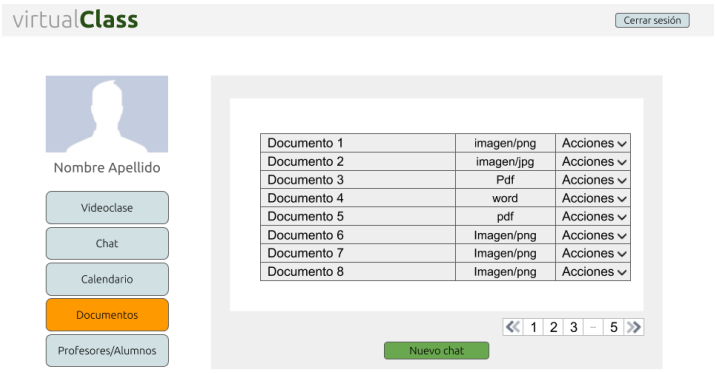
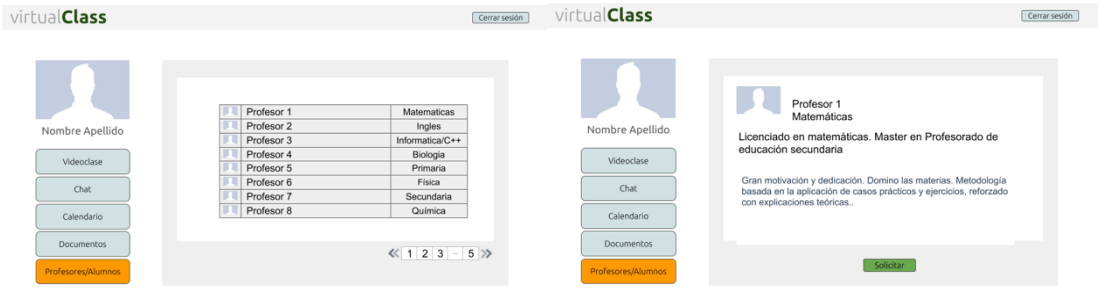


Figura 8. Diseño de la vista del apartado Documentos.

3.7. Diseño del apartado Profesor/Alumno

Al seleccionar en el menú la opción Profesor aparecería en esta interfaz un listado con los diferentes usuarios de la plataforma y la asignatura que imparte. Al seleccionar uno de ellos la interfaz cambia por el perfil profesional de éste, y se puede seleccionar para enviar una solicitud para iniciar una relación profesor - alumno a través de la plataforma.



Figuras 9 y 10. Diseño de la vista del apartado Profesor/Alumno.

4. Desarrollo del proyecto

Como se ha explicado en el presente documento, el desarrollo de este proyecto tiene como objetivo la creación de una plataforma web que permita a profesores particulares conectarse mediante videoconferencia con sus alumnos, así como proporcionar una herramienta de dibujo (Canvas) para la realización de explicaciones adicionales. A continuación, se explica cómo se ha llevado a cabo el proceso de desarrollo de dicha plataforma.

4.1. Arquitectura del sistema

Desde un principio se ha desarrollado la aplicación pensando en la escalabilidad, la reutilización y mantenimiento del mismo. La arquitectura inicial es cliente-servidor.

El **cliente** es una aplicación web (*single plage*, en *vuejs*), y el **servidor** es un conjunto de microservicios el cual puede tener múltiples instancias de cada microservicio. Cada microservicio es independiente, contiene sus casos de uso, apis y sus eventos.

4.1.1. Diseño de microservicio

Cada servicio está diseñado siguiendo el patrón *hexagonal architecture* o *clean architecture*. De esta forma dividimos la lógica de negocio de los casos de uso y de la infraestructura. Si en un futuro se decide cambiar de base de datos de *MongoDB* a *Redis* (por ser esta última más rápida), sólo habría que añadir la clase de infraestructura que estará acoplada a una interfaz.

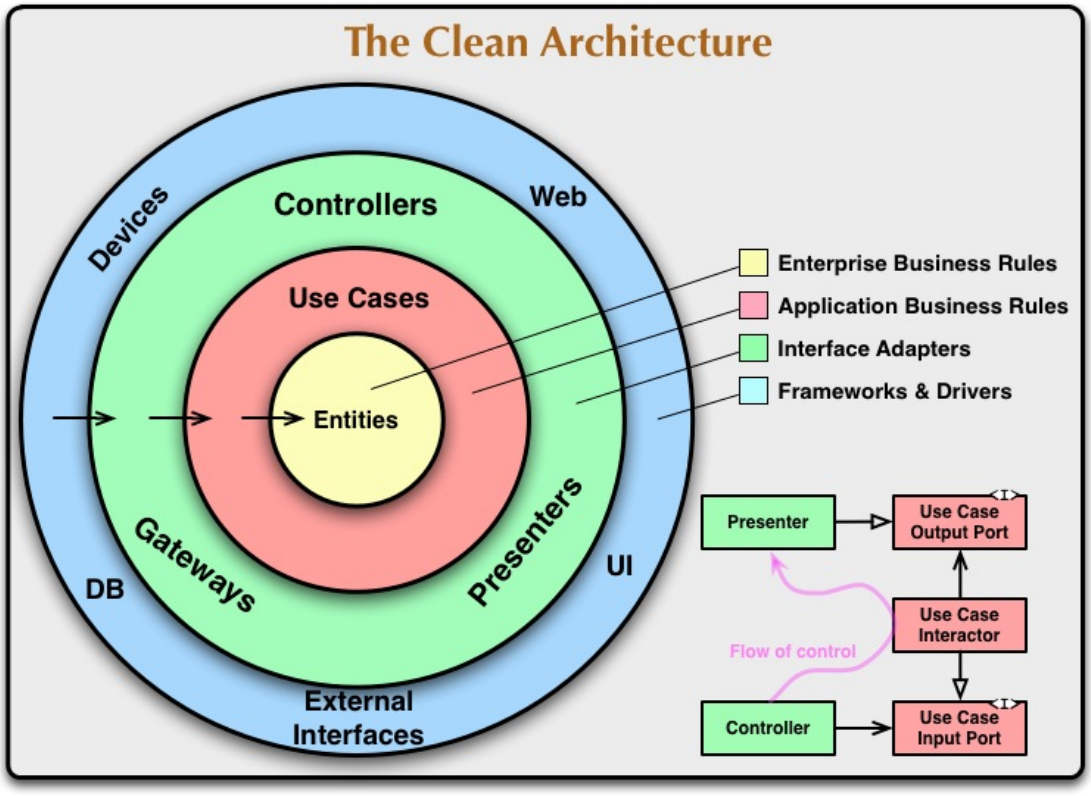


Figura 11. Arquitectura hexagonal (o Clean Architecture)

4.1.2. Comunicación entre servicios

La comunicación entre servicios se realiza mediante un sistema de colas con *RabbitMQ pub/sub*. Cada servicio tiene una única cola, a excepción del de *WebSocket* que tiene una cola por instancia. Cuando un servicio emite un evento lo emite al *exchange* de *RabbitMQ* de tipo *fanout* (a todas las colas).

Cuando un evento llega a la cola de un microservicio, *RabbitMQ* se lo da a la instancia del microservicio por medio de *round-robin*, que permite ir distribuyendo la carga. Si la ejecución del evento se ha realizado correctamente, se devuelve un *Ack*, para que *RabbitMQ* elimine el evento de la cola. En el caso del servicio *WebSocket* está puesto *noAck*, es decir, que siempre borra el evento de la cola, ya que si el servidor *WebSocket* falla y se tiene que reiniciar la conexión con el cliente es diferente y no se podrá emitir, y el cliente ya sincronizará a la reconexión.

4.1.3. Comunicación cliente-servidor

Síncrona: La comunicación entre el cliente y los microservicios se ha realizado por *Nginx* con un sistema de *reverse proxy*. Es decir, cuando llega una petición a *Nginx*, *Nginx* busca el patrón y lo reenvía al microservicio. Con *Nginx* se puede tener varias instancias apuntando al mismo patrón, de esta forma *Nginx* hará *round-robin* e irá distribuyendo la carga.

Asincrona*:* La comunicación asíncrona se realiza con *WebSocket*, el cliente establece una conexión *WebSocket* con el microservicio *WebSocket*. Cuando un microservicio externo al del *WebSocket* quiere emitir un evento, envía un evento de nombre *websocket.emit* al *RabbitMQ* y todos los servicios de *WebSocket* emiten el evento a la sala del usuario. He creado un sistema de salas para que el cliente se suscriba: primero solicita permisos al servicio especificado y a continuación le solicita al *WebSocket* añadirse a la sala enviándole la autorización (*jwt*). De esta forma es sencillo gestionar las comunicaciones externas que se desean recibir: el cliente sólo necesita suscribirse a la sala para realizar la acción requerida.

4.1.4. Diseño de la arquitectura

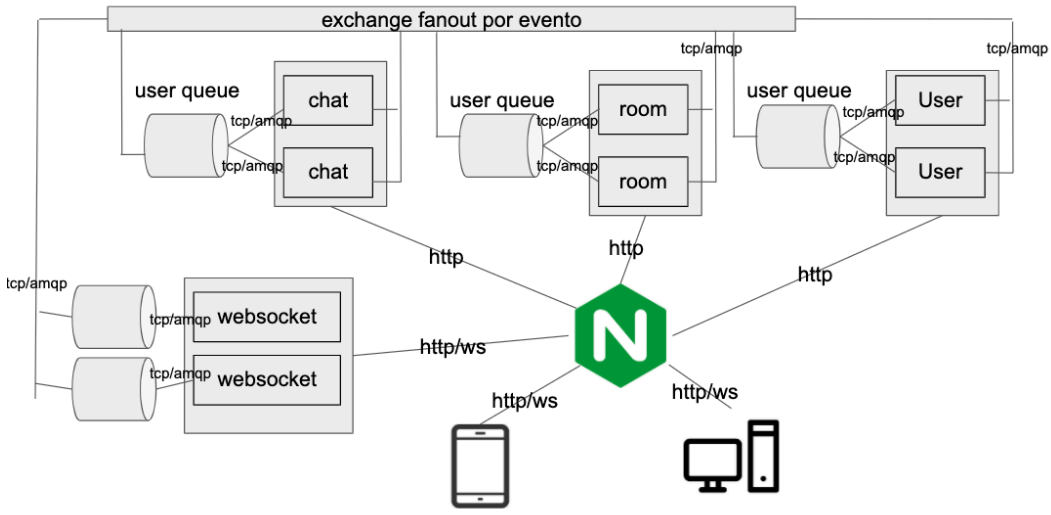


Figura 12. Diseño de la arquitectura del sistema.

4.1.5. Despliegue de la arquitectura

Cada servicio está empaquetado en contenedores *Docker* con su *Dockerfile*. La configuración de cada microservicio está en *Docker Compose,* con sus instancias de base de datos y los puertos que escucha y recibe. Con *Docker Compose* los servicios se levantan manualmente, pero también existe la posibilidad de utilizar *Kubernetes* o *Docker swarm* para ello, dado que son orquestadores de contenedores y de esta forma van desplegándose instancias según necesidades.

4.2. Servicios

4.2.1. Usuario

Este servicio es el que se encarga de registrar a profesores y alumnos, y comunicarlos entre ellos.

- **USER**
  - Entitities
    - User
      - id
      - name
      - surname
      - email
      - roles
    - Teacher extend User
      - students
      - subjects
      - degrees
      - description
    - Student extend User
      - teachers
  - Events/Async
    - event:userCreated
    - event:userUpdated
    - event:userAssigned
    - websocket:userAccepted
    - websocket:userRequested
  - API
    - user
      - get
      - put
    - users
      - get
    - teacher
      - get
      - post
      - put
    - teachers
      - get
    - studend
      - get
      - post
      - pust

        - students
            - get

4.2.2. Videollamada y Canvas

Este servicio crea salas para comunicarse por videollamada, chats y canvas.

- **CALL**
    - Entities
        - Room
            - id
            - users
            - status
            - session
            - tokens
        - RoomHistory
            - type
            - status
            - date
    - Events
        - roomCreated
        - roomUserConnected
        - roomUserDisconnected
    - API
        - room
            - get
            - post
        - rooms
            - get
- **DRAWCANVAS**
    - Entities
        - Canvas
            - id
            - owner
            - shared
            - lines
                - name
                - color
                - shadow
                - length
                - point
    - Events
        - lineCreated
    - Api
        - canvas
            - get
            - post
        - line
            - post
        - lines
            - get

4.2.3. Chat

Este servicio se encarga de crear chats y enviar y recibir mensajes

- **CHAT**

- Entitities
  - Chat
    - id
    - type
    - users
    - lastMessages
    - unreadMessagesUser
  - Message
    - id
    - chat
    - type
    - value
    - sender
    - createdAT
- Events
  - event: chatCreated
  - event: messageCreated
  - event: messageReaded
  - websocket: /chat/message
  - websocket: /chat/message/readed
- API
  - chat
    - get
    - post
  - chats
    - get
  - message
    - post
  - messages
    - get

4.2.4. WebSocket

Este servicio sirve para comunicar el cliente con el servidor de forma asíncrona por medio de salas

- **WEBSOCKET**
  - Entities
    - websocket
    - rooms
  - Events:
    - socketUserConnected
    - socketUserDisconnected

4.2.5. Documentos

Se encarga de gestionar la subida y descarga de documentos, así como compartirlos entre usuarios.

- **DOCUMENT**
  - Entitities
    - Document
      - id
      - name
      - path

        - type
        - owners
        - shared
    - Events
        - documentCreated
        - documentShared
    - API
        - document
            - get
            - post
        - documents
            - get
        - share
            - post

4.2.5. Frontend

Aplicación *single-page* del cliente.

El *frontend* está diseñado sobre *vuejs*, utilizando vue-router para cambiar de rutas y *vuex* como almacenamiento en memoria. Cada vista tiene su fichero *.vue* y reutiliza componentes compartidos. Externamente a *vue* he creado una carpeta con las interfaces y repositorios para conectarse con la api.

4.3. Vistas finales

Figura 13. Vista final de la web principal

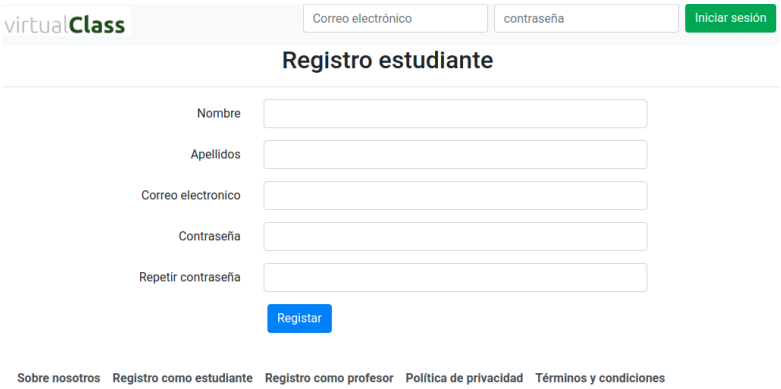


Figura 14. Vista final del registro como alumno

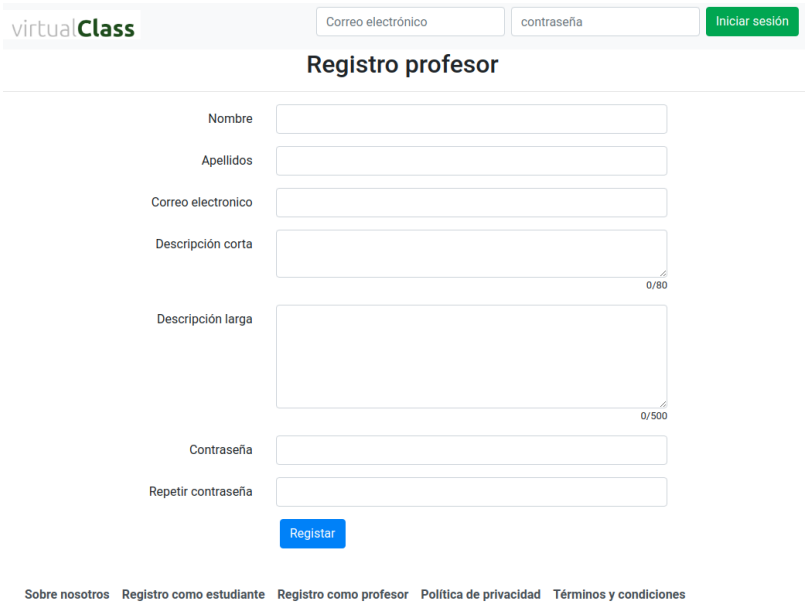
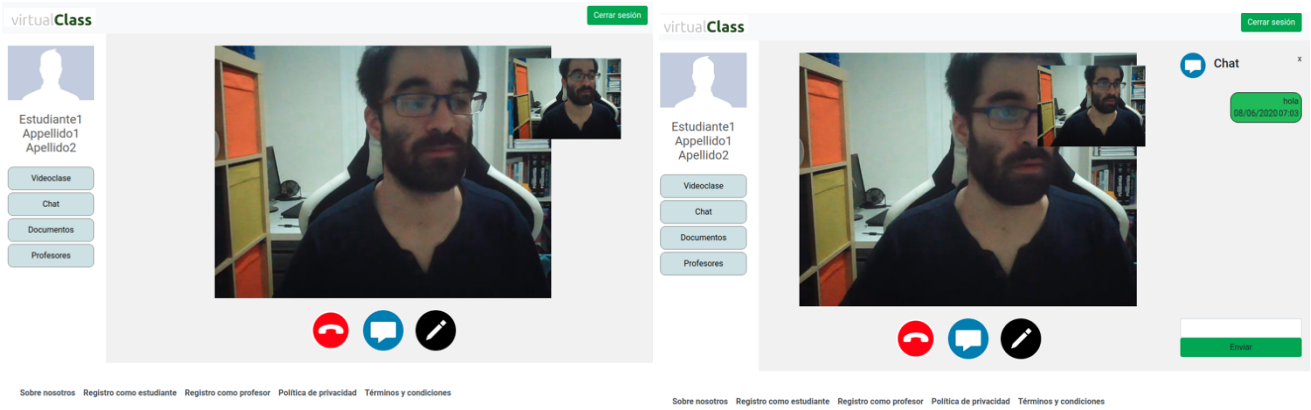
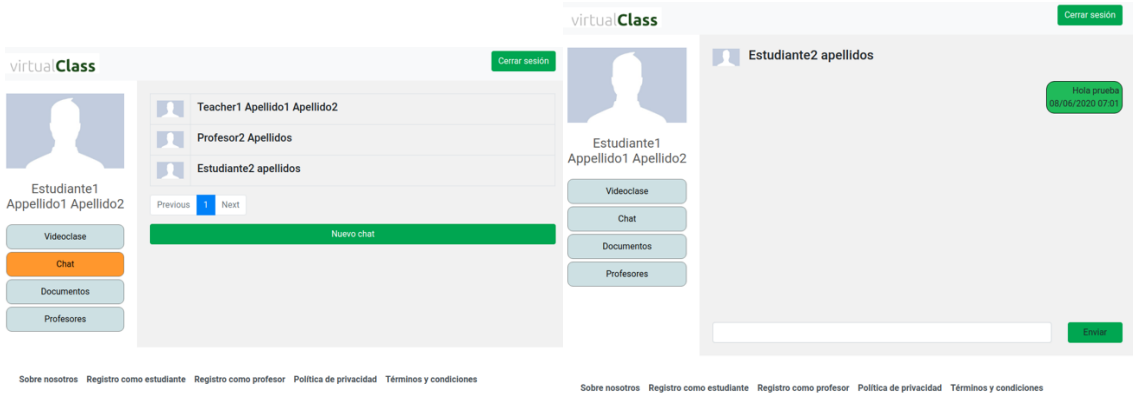


Figura 15. Vista final del registro como profesor



Figuras 16 y 17. Vistas finales de la videoclase



Figuras 18 y 19. Vistas finales del chat

4.4. Código

El código del proyecto se puede encontrar en el siguiente repositorio GitHub: https://github.com/monolo/virtualclass

5. Fase de pruebas

La fase de pruebas de una aplicación o página web es esencial a la hora de lograr un correcto funcionamiento de la misma. Los objetivos principales son lo de encontrar y documentar defectos que pueden darse a lo largo del desarrollo de la aplicación web para poder ser corregido, así como validar que funciona para lo que se ha diseñado.

He creado test unitarios en las entidades de dominio y aplicación. Además, testear los eventos asíncronos es más complicado y hay casos en los cuales ha sido más dificultoso ver la causa. Las *apis* las he ido testeando con la aplicación *postman*, que te permite enviar peticiones http con facilidad.

A continuación, muestro capturas de las pruebas en la terminal:

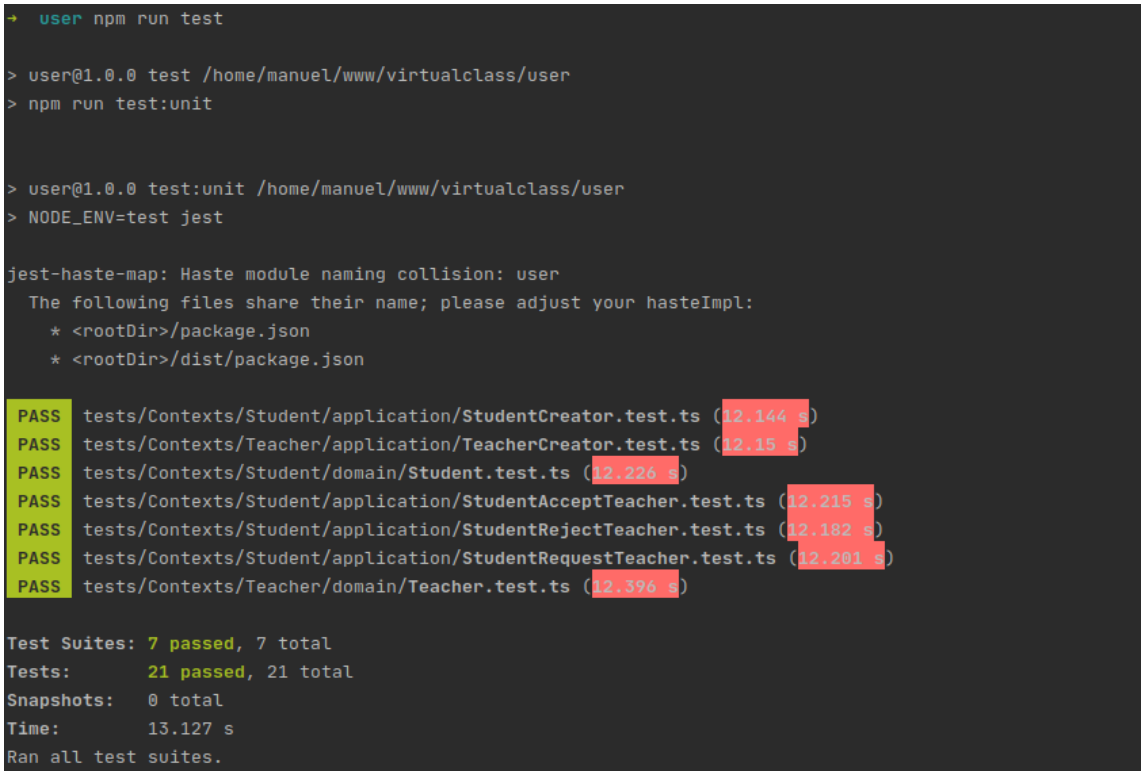


Figura 20. Test unitario de usuario

Por otro lado, se ha analizado el acceso desde diferentes navegadores y dispositivos (ordenadores, tabletas y teléfonos móviles). La aplicación web se había desarrollado con la intención de un adecuado funcionamiento en todos los navegadores, lo cual quedó corroborado al realizar las pruebas pertinentes.

Para finalizar la fase de pruebas se ha un *test* con una tercera persona ajena al desarrollo de la aplicación, quien debía poder realizar las siguientes tareas: registro como usuario, acceder a la plataforma, realizar videollamada, utilización del chat y subir un documento. No se ha podido considerar el *test* como exitoso al haber surgido un error en al subir documentos a la plataforma; por falta de tiempo este error no se ha podido solucionar, por lo que finalmente se ha eliminado de las funcionalidades de VirtualClass. El resto de las tareas (registro, acceso, videollamada y chat) se han podido realizar sin problemas.

6. Conclusiones

Al concluir el proyecto se cumplió con el principal objetivo del trabajo: diseñar y desarrollar un sitio web que permitiese la realización de videoclases entre profesores y alumnos. Sin embargo, los objetivos planteados inicialmente no se han podido cumplir con la precisión que me hubiera gustado (al no haber podido incluir el Canvas, el Calendario y la opción de Documentos), ya que la complejidad de la arquitectura asíncrona y distribuida ha supuesto más problemas que beneficios para un proyecto como este. En un proyecto pequeño con una lógica de negocio que no requiera una gran escalabilidad no recomendaría este tipo de infraestructura.

En cuanto al diseño, se logró una experiencia de usuario óptima y sencilla, con un diseño acorde al planteado en un principio. El diseño web adaptable (*responsive*) dio algún problema en dispositivos pequeños (como teléfonos móviles) que finalmente pudieron ser solucionados.

El desarrollo de la plataforma fue quizás la parte más complicada, teniendo que realizar numerosas pruebas y cambios sobre la marcha, y habiendo tenido que eliminar algunas de las funcionalidades que se tenían previstas implementar (por ejemplo: la opción del calendario y la de documentos).

Al desarrollar el proyecto he aprendido a gestionar mensajes de colas y comunicación asíncrona en sistemas distribuidos. Los desarrollos en sistemas distribuidos son complejos y añaden una capa de complejidad elevada respecto a los proyectos monolíticos en un único servicio, pero aportan grandes beneficios de escalabilidad y de automatización de tareas en segundo plan, y por ello sin estos sistemas no podrían existir las webs complejas actuales.

Inicialmente se desarrolló correctamente siguiendo la planificación inicial (modelado de modelos y casos de uso), pero cuando fue siendo más necesaria la comunicación asíncrona y distribuida se fue retrasando la planificación, al tener que dedicar más tiempo del planificado para intentar cumplir con los objetivos e ir reinscribiendo servicios para ajustarlos a la arquitectura.

Por otra parte, faltaría explorar mejor la escalabilidad con múltiples instancias de cada servicio, orquestar los servicios con *Kubernestes* o *Docker Swarm*, monitorizar y realizar *logs* de cada servicio y evento.

Una de las grandes dificultades que he encontrado durante el desarrollo de este proyecto ha sido tener que afrontar todas las fases del proyecto un mismo individuo. Esto dificulta la visión de errores, mientras que en un equipo de varias personas estos errores podrían haber sido evidentes de forma más temprana.

En conclusión, se podría decir que esta aplicación tiene un gran potencial de desarrollo y un futuro prometedor. En cuanto al diseño de éste, se puede concluir que cumple con las condiciones necesarias para facilitar su usabilidad por parte del usuario. Y estoy contento con lo muchos que aprendido desarrollando el proyecto y he salido unos grandes conocimientos que seguro que utilizaré en un futuro.

7. Glosario

- **Canvas**: región dibujable definida en el código HTML, que permite la generación de gráficos dinámicamente. [21]
- **Sprint**: intervalo prefijado durante el cual se crea un producto utilizable y/o potencialmente entregable. Es desarrollo de un proyecto está constituido por diferentes *sprints*.
- **Framework**: Entorno de trabajo. En el desarrollo de software, un entorno de trabajo es una estructura conceptual y tecnológica de asistencia definida que puede servir de base para la organización y desarrollo de *software*. Puede incluir soporte de programas, biblioteca, entre otras herramientas, para así ayudar a desarrollar los diferentes componentes de un proyecto.
- **Frontend y Backend**: son términos que se refieren a la separación de intereses entre una capa de presentación y una capa de acceso a datos, respectivamente. En diseño de software el *frontend* es la parte del software que interactúa con el usuario, mientras que el *backend* es la parte que se encarga de recolectar procesar la entrada de datos.
- **Contenedor Docker**: Docker es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de virtualización de aplicaciones en múltiples sistemas operativos. [6]
- **Websocket**: es una tecnología que proporciona una canal de comunicación bidireccional. Está diseñada para ser implementada en navegadores y servidores web, pero puede utilizarse en cualquier aplicación cliente/servido.
- **Api Gateway**: es una herramienta de gestión de API, la cual toma todas las llamadas de la API de los clientes y las encamina al microservicio apropiado con enrutamiento de solicitudes, composición y traducción de protocolos; de este modo determina qué servicios se necesitan y los combina en una experiencia sincrónica para el usuario.
- **WebRTC**: es un proyecto libre y de código abierto que proporciona a los navegadores web y a las aplicaciones móviles comunicación en tiempo real (RTC) a través de interfaces de programación de aplicaciones (APIs). Permite que la comunicación de audio y vídeo funcione dentro de las páginas web al permitir la comunicación entre pares, eliminando la necesidad de instalar *plugins* o descargar aplicaciones nativas. [21,22]
- **Responsive** (diseño web adaptable): diseño y desarrollo cuyo objetivo es adaptar la apariencia de las páginas web al dispositivo que se esté utilizando para visitarlas (ordenador, tableta o teléfono móvil).

8. Bibliografía

1. *Linux.org.* [Consultado el 14 de abril de 2020]. Disponible en: https://www.linux.org/
2. *Nginx.com.* [Consultado el 14 de abril de 2020]. Disponible en: https://www.nginx.com/
3. Chodorow K, Diroff M. *Mongo DB: The definitive guide*. O'Reilly Media; 2011.
4. *Redis.io*. [Consultado el 3 de mayo de 2020]. Disponible en: https://redis.io/
5. Richard E. Silverman. *Git Pocket Guide: A Working Introduction.* 1ra Edición. Editorial O'Reilly Media; 2013.
6. Steven J. Vaughan-Nichols. *Docker libcontainer unifies Linux container powers.* Publicado 11 de junio de 2014; [Consultado el 5 de junio de 2020]. Disponible en: https://www.zdnet.com/article/docker-libcontainer-unifies-linux-container-powers/
7. Ornbo G. *Programación Node.js.* Editorial Anaya; 2013.
8. *Express*. [Consultado el 23 de abril de 2020]. Disponible en: https://expressjs.com/es/
9. *Socket.io*. [Consultado el 3 de mayo de 2020]. Disponible en: https://socket.io/
10. *Mongoose*. [Consultado el 23 de abril de 2020]. Disponible en: https://mongoosejs.com/
11. Rubiales Gómez M. *Curso de Desarrollo Web: HTML, CSS y JavaScript*. Editorial Anaya; 2018
12. *Learn to style HTML using CSS.* Publicado 28 de abril de 2020; [Consultado el 15 de mayo de 2020]. Disponible en: https://developer.mozilla.org/en-US/docs/Learn/CSS
13. Sawyer D. *JavaScript y jQuery*. Editorial Anaya; 2012
14. Hassan D, Murray N, Ari L. *Fullstack Vue: The Complete Guide to Vue.js.* 1ª Edición. Editorial CreateSpace Independent; 2018.
15. Ángel Álvarez M. *Librería Axios: cliente HTTP para Javascript.* Publicado 14 de septiembre de 2018; [Consultado el 18 de abril de 2020]. Disponible en: https://desarrolloweb.com/articulos/axios-ajax-cliente-http-javascript.html
16. Tomás E. *Qué es REST.* Publicado 25 de abril de 2014; [Consultado el 18 de abril de 2020]. Disponible en: https://desarrolloweb.com/articulos/que-es-rest-caracteristicas-sistemas.html
17. Wang V, Salim F, Moskovits P. *The Definitive Guide to HTML5 WebSocket.* 1ª edición. Editorial Apress; 2013.
18. *Rabbitmq.com* [Consultado el 13 de abril de 2020]. Disponible en: https://www.rabbitmq.com/
19. *How WebRTC Is Revolutionizing Telephony*. Publicado 21 de febrero de 2014; [Consultado el 29 de mayo de 2020]. Disponible en: https://blogs.trilogy-lte.com/post/77427158750/how-webrtc-is-revolutionizing-telephony
20. WebRTC [Consultado el 29 de mayo de 2020]. Disponible en: https://webrtc.org/
21. Campos Ó. Introducción al elemento canvas de HTML5. Publicado 20 de junio de 2011. [Consultado el 7 de mayo de 2020]. Disponible en: https://www.genbeta.com/desarrollo/introduccion-al-elemento-canvas-de-html5