UNIVERSIDAD POLITÉCNICA DE MADRID

Escuela Técnica Superior de Ingenieros de Telecomunicación

CONTRIBUCIÓN A UNA HERRAMIENTA WEB DE AUTORÍA DE RECURSOS DE E-LEARNING MEDIANTE LA INTEGRACIÓN DE UN COMPONENTE DE REALIDAD VIRTUAL

GRADO EN INGENIERÍA DE TECNOLOGÍAS Y SERVICIOS DE TELECOMUNICACIÓN

Trabajo Fin de Grado

PAULA GONZÁLEZ GÓMEZ

Madrid, España
Julio, 2018

Departamento de Ingeniería de Sistemas Telemáticos

Escuela Técnica Superior de Ingenieros de Telecomunicación

CONTRIBUCIÓN A UNA HERRAMIENTA WEB DE AUTORÍA DE RECURSOS DE E-LEARNING MEDIANTE LA INTEGRACIÓN DE UN COMPONENTE DE REALIDAD VIRTUAL

GRADO EN INGENIERÍA DE TECNOLOGÍAS Y SERVICIOS DE TELECOMUNICACIÓN

Trabajo Fin de Grado

Autor: PAULA GONZÁLEZ GÓMEZ

Tutor: Lourdes Marcos Pascual

Ponente: Enrique Barra Arias

Julio, 2018

Presidente: _____

Vocal: _____

Secretario: _____

Suplente: _____

Realizado el acto de defensa y lectura del TFM el día _____ de _____ de _____ en Madrid, habiendo obtenido la calificación de _____

El presidente, El secretario, El vocal,

Resumen

Ediphy es una herramienta web de código libre para la creación de recursos educativos que se estructura en base a módulos llamados plugins. A través de un entorno de edición y la agregación de dichos plugins, los usuarios pueden crear y previsualizar los recursos creados.

El objetivo que persigue este Trabajo Fin de Grado es el de ampliar las funcionalidades de Ediphy mediante la integración de un plugin de realidad virtual. La nueva funcionalidad ofrece a los usuarios la posibilidad de crear entornos 360ž con la imagen que seleccionen, habilitar sonido ambiente, y añadir un carrusel de imágenes en la escena. Además el plugin permite añadir puntos calientes (marcas) en el espacio tridimensional con los que poder incluir mensajes emergentes con contenido explicativo o navegación entre las vistas del recurso.

Para el desarrollo del trabajo se planteó una primera fase de investigación sobre las librerías de realidad virtual y sus posibles aplicaciones en el contexto de Ediphy. La fase siguiente de desarrollo se centró en establecer una comunicación efectiva entre la aplicación de realidad virtual y Ediphy que permitiese la configuración del entorno virtual. Una vez establecida la comunicación, se realizaron tanto la implementación de los módulos nativos de la aplicación de realidad virtual, como la implementación del plugin y su configuración en Ediphy. Finalmente, la fase de validación permitió comprobar la correcta integración de la nueva funcionalidad agregada.

Palabras clave: Ediphy, plugins, multimedia, realidad virtual, e-Learning

Abstract

Ediphy is an open source web-based platform for the creation of educational resources. Its structure is based on modules named plugins, which users can aggregate in an editing environment to generate and preview the resources they are creating.

The goal of this Bachelors Thesis is to extend the functionality of Ediphy by integrating a virtual reality plugin. This new functionality allows users to create 360-degree environments featuring the desired images, ambient sound, and a carousel of images. The plugin also allows the creation of hotspots (marks) in the three-dimensional space that can be used to display pop-up messages or help navigate the resource.

The first phase of the development involved researching the state of the art of software libraries for virtual reality and their potential for the context of Ediphy. The next phase focused on establishing a communication link between the virtual reality application and Ediphy that allowed for the configuration of the virtual environment. Once this connection was established, we implemented the native modules within the virtual reality application, as well as the plugin and its configuration in Ediphy. Finally, the validation phase allowed us to verify the correct integration of the newly added functionality.

Keywords: Ediphy, plugins, multimedia, virtual reality, e-Learning

Agradecimientos

A mi tutora, Lourdes, por haberme dicho convencida que sí se podía y ayudarme a conseguirlo.

A Sonsoles, por estar siempre disponible y saber resolver cualquier duda.

A mis padres, por haber mantenido la confianza en mí por encima de todo y responder al teléfono siempre con una sonrisa y todo el ánimo del mundo.

A mi hermana, por ser la persona que mejor me conoce y su imposición del pensamiento positivo.

A mi eterno compi de la uni, Julián, porque somos un equipo.

Y a mi abuela, por tener siempre encendida una velita.

Gracias.

Índice

ÍNDICE

Ilustraciones

ILUSTRACIONES

Tablas

Acrónimos

API Interfaz de Programación en Aplicaciones

DOM Modelo de Objeto de Documento

ECMA European Computer Manufacturers Association

ES6 ECMAScript

GPU Graphics Processing Unit, Unidad de Procesamiento Gráfico

JS JavaScript

MVC Modelo Vista Controlador

PC Personal Computer, Ordenador Personal

RV Realidad Virtual

TIC Tecnologías de la Información y la Comunicación

UPM Universidad Politécnica de Madrid

URI Uniform Resource Identifier, Identificador Uniforme de Recursos

URL Uniform Resource Locator, Localizador Uniforme de Recursos

WebGL Web Graphics Library

WebVR Web Virtual Reality

3D Tres Dimensiones

2D Dos Dimensiones

Capítulo 1

Introducción y objetivos

La educación en los últimos veinte años ha sufrido una transformación liderada por las Tecnologías de la Información y la Comunicación (TIC) que ha cambiado la manera en la que se imparten los contenidos educativos, afectando a varias esferas, desde la inclusión de contenidos y materias relacionadas con las TIC, hasta la forma en la que se transmiten dichos contenidos.

Uno de los términos más importantes en esta transformación es el de formación en línea o enseñanza virtual, que se conoce generalmente por su expresión en inglés *e-Learning*. Este concepto se refiere a los procesos de enseñanza y aprendizaje que se llevan a cabo mediante el uso de Internet, y que se caracterizan por la separación física entre profesores y alumnos. Debido a su flexibilidad, este tipo de aprendizaje ofrece al alumno la posibilidad de autogestionar el ritmo de aprendizaje, y le permite optimizar el tiempo dedicado a la formación. Según el análisis que hacen Valverde, López, Garrido, Díaz y Rosenberg [1], la enseñanza virtual ofrece numerosas ventajas como son la mayor riqueza del proceso formativo, aumento de motivación por el aprendizaje, la comunicación entre los agentes educativos, el seguimiento del proceso de aprendizaje, facilidad de actualización de contenidos, reducción de costes y fácil acceso.

En este escenario, resulta cada vez más adecuada la incorporación de nuevas herramientas y tecnologías que permitan la creación de recursos educativos dinámicos, que propongan formas alternativas de aprendizaje ajustadas a las características de los nuevos soportes. La realidad virtual, por sus cualidades de inmersión, se muestra como una posible solución para ampliar las funcionalidades del *e-Learning*. Como ya hemos comentado, una de las mayores ventajas que ofrece este tipo de educación es la de desagregar el espacio y el momento temporal de la enseñanza por parte del profesor hacia el alumno. Por eso, la posibilidad de construir espacios tridimensionales que puedan llegar a ser consumidos en visores especializados y en cualquier momento o lugar, abre un nuevo abanico de posibilidades para el desarrollo de la educación a través de las TIC.

El objetivo principal de este trabajo se centra en ampliar las funcionalidades de Ediphy, una herramienta web de autor y código libre cuyo propósito general es la creación y edición de recursos educativos multimedia que ayuden a la generación de contenido educativo de calidad para su uso en *e-Learning*. Dicha mejora se focaliza en permitir a los usuarios utilizar la realidad virtual como soporte para la generación de contenido educativo.

La estructura modular en la que se basa Ediphy facilita la contribución al desarrollo con nuevas funcionalidades que se pueden agregar de forma sencilla mediante la creación de unidades mínimas de contenido a las que denomina *plugins*. Aprovechando la facilidad que ofrece la herramienta, se ha incorporado un nuevo *plugin* que permite añadir escenarios de realidad virtual a los autores de los recursos creados.

Para alcanzar el objetivo que se plantea, este trabajo se ha desarrollado en varias fases, las cuales han ido cumpliendo distintos objetivos para llegar al resultado final. Durante la primera fase de investigación se han estudiado las librerías de realidad virtual disponibles y las posibles aplicaciones que estas podían tener en un contexto como Ediphy. A continuación, se ha diseñado un modelo de comunicación entre los componentes de la aplicación de realidad virtual y el *plugin* de Ediphy que permitiese configurar desde la barra de herramientas el entorno virtual. Con este objetivo cumplido, se ha podido empezar a desarrollar tanto la implementación de los módulos de realidad virtual como la de los aspectos relacionados con la configuración del *plugin* en Ediphy. Por último, durante la fase de validación se ha comprobado la correcta integración de la nueva funcionalidad agregada.

El nuevo *plugin* de Ediphy permite a los usuarios elegir entre los entornos disponibles, tanto en repositorios especializados como de una selección local. En estos entornos, los usuarios podrán crear paseos de aprendizaje utilizando marcas con información posicionadas a lo largo de todo el mundo virtual seleccionado. Los resultados que ofrece este tipo de inmersión a la hora de retener conocimiento son indudablemente mejores que con métodos tradicionales, pues ayudan a la concentración y asociación de conceptos por parte del alumno.

Capítulo 2

Estado del arte

2.1. Tecnologías

2.1.1. JavaScript

JavaScript (JS) [2] [3] es un lenguaje ligero, interpretado y basado en prototipos. Fue diseñado por Brendan Eich en 1995 y actualmente es una marca registrada de Oracle Corporation. Se utiliza en el desarrollo de páginas web dinámicas para dar respuestas rápidas a las interacciones del usuario, sin necesidad de enviar al servidor todas las acciones y tener que esperar por la respuesta, ya que se ejecuta en el cliente. JavaScript sigue el estándar *ECMAScript*[4], publicado por la European Computer Manufacturers Association (ECMA), y ha ido evolucionando a través de diferentes versiones compatibles con las anteriores. En 2015, se publicó *ECMAScript 6* (ES6), que proporciona un mejor soporte para aplicaciones complejas, permite la creación de bibliotecas y el uso de *ECMAScript* como objetivo de compilación para otros lenguajes. Las mejoras de esta nueva versión del estándar están relacionadas con los módulos *import* y *export* (que permiten a los analizadores estáticos construir un árbol completo de las dependencias sin ejecutar código), declaraciones de clases, iteradores, generadores y promesas de programación asíncrona, entre otros aspectos. ES6 proporciona además la base para mejoras regulares e incrementales de lenguaje y bibliotecas, lo que ha propiciado que el estándar *ECMAScript* entre en ciclos de lanzamiento anuales. El siguiente lanzamiento, en cuyo borrador se está trabajando actualmente es el de *ECMAScript 2019*. Al igual que lenguajes como Java o C, con JavaScript se pueden almacenar valores dentro de variables, implementar funciones, utilizar operadores sobre objetos estándar o tipos nativos, etc. Sin embargo, una de las funcionalidades más importantes de este lenguaje es la denominada Interfaz de Programación en Aplicaciones (API). Este tipo de interfaces basan su funcionamiento en la inserción de código listo para ser reutilizado en diversas aplicaciones, facilitando la labor a los desarrolladores. Las API se pueden clasificar en dos categorías: de

navegadores y de terceros. La primera incluye aquellas API que se construyen en el navegador web y que exponen información referente al propio dispositivo, como el Modelo de Objeto de Documento (DOM), la geolocalización, *Canvas* o *Web Graphics Library* (WebGL). Por su parte, las API de terceros serían por ejemplo la de Twitter [5], que permite obtener *tweets* de algún usuario y mostrarlos en una web propia, o la de Google Maps [6], con la que se pueden incrustar mapas personalizados entre otras funcionalidades. Se trata, como ya se ha comentado, de un lenguaje con una gran presencia en las páginas web de hoy en día, que no se compila para poder ejecutarse, sino que es interpretado y soportado por todos los navegadores actuales (toda la lógica en el cliente). Utiliza prototipos para la herencia en lugar de clases (diferencia importante con Java) y es orientado a objetos y eventos.

2.1.2. React

Se trata de una librería JavaScript [7] de código abierto, desarrollada por Facebook y lanzada en 2013, cuyo objetivo es facilitar a los desarrolladores la creación de interfaces de usuario interactivas. Actualmente las aplicaciones alcanzan en muchas ocasiones un nivel de complejidad que exige que la estructura en la que se basan esté bien organizada, sea flexible para adaptarse a los cambios y pueda ser reutilizable en su totalidad o por partes. Esta idea de dividir en módulos las aplicaciones es la que anima a los desarrolladores a utilizar arquitecturas basadas en microservicios, componentes o módulos. Siguiendo con esta tendencia, la librería de Facebook fue precisamente diseñada para basarse en el uso de componentes, los cuales poseen un estado del que depende lo que representan en cada momento. Ayudan a redistribuir la complejidad de la aplicación global, encargándose de una tarea más simple dentro de la funcionalidad total de manera que, según sus datos de entrada (en adelante *props*) y su estado interno, devuelven HTML con el contenido que corresponda. Son fáciles de testear mediante el uso de test unitarios, se declaran con JSX[8] y deben escribirse en ES6. Además, es importante conocer el ciclo de vida de estos elementos a la hora de desarrollar aplicaciones con React, pues de ello depende que el comportamiento de la misma sea en el orden y con el resultado esperado.

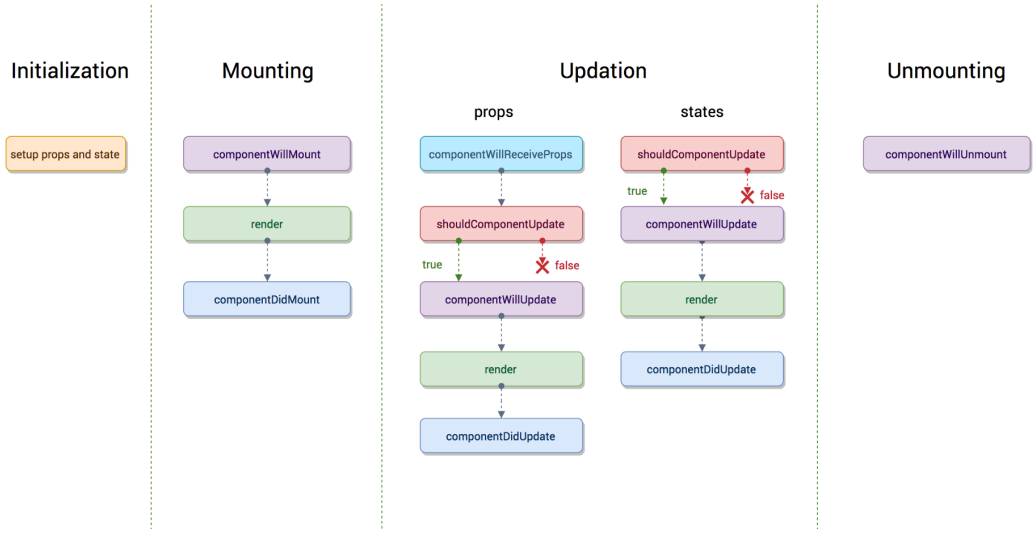

Figura 1 : Ciclo de vida de los componentes React

Existen más métodos opcionales que se pueden implementar a parte de los mostrados en la figura anterior, pero el *render* es obligatorio y debe devolver un único elemento (ya sea un componente DOM o uno React). Otra de las características de React es que utiliza un flujo de datos unidireccional en el patrón Modelo Vista Controlador (MVC). Así, los eventos suben hacia los componentes más complejos y su estado controla la aplicación, mientras que los datos bajan hacia los componentes más simples.

Un término importante dentro de la librería de React es el de DOM virtual. Se trata de una copia del DOM del navegador que guarda React y se utiliza para determinar qué partes del mismo han sido modificadas comparando la versión que se va a renderizar con la anterior. De esta manera, se actualiza eficientemente el DOM del navegador.

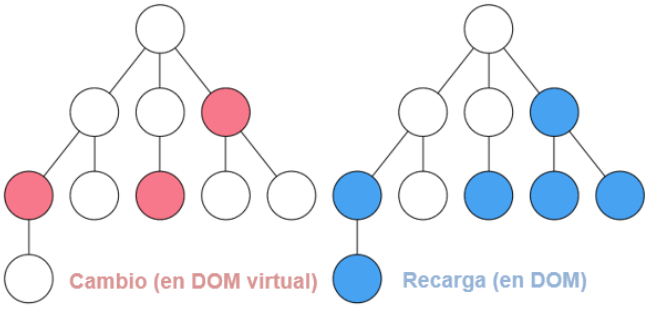

Figura 2 : Concepto de DOM virtual

2.1.3. Redux

Redux [9] es una librería que implementa el patrón de diseño Flux [10], aunque con algunas variaciones. Es un contenedor predecible del estado de aplicaciones JavaScript y ayuda a escribir aplicaciones consistentes y fáciles de probar. Antes de entrar más en detalle sobre cómo funciona Redux, cabe destacar que el patrón o arquitectura Flux se utiliza en Facebook para sus aplicaciones y, por lo tanto, es completamente compatible con los componentes de React, ya que también fomenta el flujo de datos unidireccional. Redux puede ser descrito en tres principios fundamentales:

- Única fuente de la verdad: el estado de toda la aplicación está almacenado en un árbol guardado en único almacén.
- El estado es de sólo lectura: la única forma de modificar el estado es emitiendo una acción (estas describen qué ocurrió).
- Los cambios se realizan con funciones puras: se utilizan *reducers* puros para especificar cómo las acciones transforman el estado de la aplicación.

Los conceptos básicos de la librería Redux son: el estado, las acciones, los *reducers* y el almacén. El estado es inmutable (no se modifica directamente) y modela el estado de la aplicación como un único objeto JavaScript. Las acciones son las únicas mediante las que se puede modificar el estado. Son también objetos JavaScript y describen el cambio a realizar. Se llaman acciones porque se invocan debido a las interacciones del usuario con la aplicación. Los *reducers* son funciones puras (sólo utilizan parámetros de entrada, no recurren a ningún elemento fuera de ellas) que aplican acciones sobre el estado. Toman como parámetros el estado anterior y una acción, y devuelven el estado debidamente modificado. El almacén reúne el estado y los *reducers* para permitir que estos últimos accedan y actualicen el estado, registra los *listener* y maneja la anulación de este registro vía el retorno de la función *subscribe(listener)*.

Para utilizar Redux en una aplicación React se tiene que transferir el estado y las acciones que lo modifican. Para ello, se crea un componente <*ReduxProvider/*> que le pasa el estado al componente raíz de la aplicación. Se puede dejar parte del estado en los componentes de React en caso de que se trate de información exclusivamente relevante para las vistas.

2.1.4. Webpack

Webpack [11] es un empaquetador de módulos estáticos para aplicaciones de JavaScript. Al procesar la aplicación, la herramienta crea un gráfico de dependencia que mapea cada módulo necesario para el proyecto y genera uno o más paquetes con la aplicación procesada.

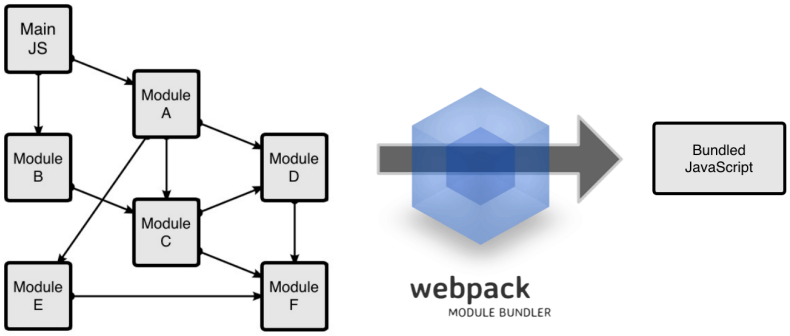

Figura 3 : Empaquetado Webpack

Una de las ventajas de Webpack es la fragmentación de código. Solamente se cargan las partes necesarias en cada petición, ya que no todos los componentes de una aplicación web requieren del código JavaScript completo. Webpack es compatible con todos los navegadores actuales (los que soportan ES5 como mínimo) y facilita el compilado de códigos en otros lenguajes a JavaScript nativo, la compresión de este o la transpilación a ES5 de estándares *ECMAScript* más avanzados, que aún no estén tan extendidos entre los navegadores. Durante el proceso de desarrollo se lanza un servidor que empaqueta todo el código y sus dependencias, transpila de forma automática los ficheros modificados y presenta la aplicación cambiada en el navegador. Para pasar el proyecto a producción, la herramienta transforma todo el código y las dependencias en un único *script (bundle)*. Hasta la versión 4, eran estrictamente necesarios uno o varios ficheros de configuración para empaquetar todo el código. Se mantenían así configuraciones separadas para el desarrollo y la producción del proyecto. Actualmente ya no son necesarios estos ficheros, ya que Webpack cuenta con una configuración por defecto. Sin embargo, sigue siendo ampliamente configurable para adaptarse a las necesidades de cada aplicación, y para eso el usuario puede crear un fichero llamado *webpack.config.js* en el directorio raíz del proyecto que la herramienta detectará y usará automáticamente. Los conceptos de configuración fundamentales son:

- *Entry point*: indica el módulo por el que Webpack debe comenzar a construir el gráfico de dependencia interno. Por defecto es *./src/index.js*
- *Output*: indica dónde deben crearse los paquetes y cómo nombrar a los archivos. Por defecto vale *./dist/main.js* (archivo de salida principal) y el resto de archivos en *./dist*
- *Loaders*: son los que permiten a Webpack procesar archivos que no sean JavaScript y convertirlos en módulos que puedan ser consumidos por la aplicación y añadidos al gráfico de dependencia.
- *Plugins*: se utilizan para ampliar las capacidades de Webpack, permitiéndole realizar tareas como la optimización de paquetes, la gestión de recursos activos (imágenes, audio, vídeos, etc.) o la inyección de variables de entorno.

2.1.5. React Native

React Native [12] es un *framework* que permite desarrollar aplicaciones nativas Android e iOS usando JavaScript y React. Funciona de la misma forma que React, utilizando componentes que al estructurarse correctamente forman una interfaz de usuario moderna para aplicaciones de este tipo de dispositivos. Al igual que React, ha sido desarrollado por Facebook y usada en sus aplicaciones además de continuar con la filosofía de código abierto. Las aplicaciones nativas son aquellas que se distribuyen a los usuarios mediante archivos binarios y deben ser instaladas en los dispositivos (no se consumen en el navegador). Con React Native se puede interactuar directamente con los módulos nativos del dispositivo móvil, así que se pueden crear aplicaciones exclusivamente para Android, para iOS o para que funcionen en ambas plataformas. En lugar de utilizar elementos HTML como se hace en React, en React Native se trabaja con componentes nativos. Así, elementos como <*div*>, <*header*> o <*article*> son equivalentes a <*View*> en React Native,y <*span*>, <*p*> o <*h1*> equivalen a <*Text*>. React Native proporciona diversas API para acceder a funcionalidades nativas, como *Alert*, *AppState*, *CameraRoll*, *Geolocation* y *StyleSheet*. Se mantienen los conceptos básicos de React de estado y *props*, uso de JSX y de componentes, DOM virtual, etc.

2.2. Realidad Virtual

Resulta complicado determinar el origen exacto de la realidad virtual (RV), ya que a pesar de que desde mediados del siglo pasado se comenzó a tratar en la ciencia ficción, no ha sido hasta estas últimas décadas que se han desarrollado dispositivos que acercan este concepto al mercado y, en los últimos años, aumentado la producción comercial debido a la curva de desarrollo tecnológico exponencial que han tenido. *La realidad virtual es una tecnología informática que genera entornos tridimensionales con los que el sujeto interactúa en tiempo real, produciéndose de esa manera una sensación de inmersión semejante a la de presencia en el mundo real* [13]. Hay un aspecto importante a la hora de hablar de esta tecnología y es el concepto de grado de inmersión. Los videojuegos tradicionales de los últimos años en los que, aunque el usuario interactúa con un mundo ficticio y creado por software, no se siente completamente inmerso en el mismo, entrarían en la categoría de realidad virtual semi-inmersiva. Existen distintos visores para consumir las aplicaciones RV. Desde una simple estructura de cartón con lentes especiales (Google Cardboard [14]) que se puede utilizar con cualquier *smartphone*, hasta visores más completos que permiten aumentar el grado de inmersión del usuario en el mundo virtual. De esta forma se puede interactuar con los objetos a través de mandos o incluso reconocer el desplazamiento del usuario. Entre estos últimos dispositivos se encuentran las Oculus Rift [15] y las Samsung Gear VR [16]. También se puede ver contenido 360 en los navegadores sin necesidad de utilizar un visor. De esta manera, se puede consumir la realidad virtual desde diferentes dispositivos, variando el grado de inmersión que consiguen. Un término que causa confusión a la hora de ser definido y comparado con la RV es de realidad aumentada. Este tipo de realidad se basa en la combinación del mundo real con el virtual, mediante software, para enriquecer con gráficos y elementos interactivos la experiencia visual.

2.3. Librerías de Realidad Virtual

2.3.1. Web GL. Web VR

Web Graphics Library (WebGL) [17] es una especificación estándar que define una API de JavaScript para la renderización de gráficos de tres dimensiones (3D) en el navegador, siempre que la Unidad de Procesamiento Gráfico (GPU) del dispositivo lo soporte. WebGL permite la aceleración hardware de la GPU y el procesamiento de

imágenes y efectos como parte del *canvas* de la web. Ligada a esta librería se encuentra también la de *Web Virtual Reality* (WebVR) [18]. En este caso, se habla de una especificación abierta que hace posible ejecutar experiencias de realidad virtual en el navegador. WebVR permite que las escenas de WebGL sean presentadas en las pantallas de los dispositivos como escenas de realidad virtual. Estas escenas se caracterizan por ser consumidas mediante un visor (*headset*) que consigue, mediante lentes de unas características concretas, presentar el contenido al usuario de manera inmersiva y con profundidad.

2.3.2. Three.js

En la historia del desarrollo software se puede encontrar una enorme variedad de *frameworks* cuyo objetivo es el de abstraer de la complejidad de las API de niveles inferiores o más primitivos, y facilitar de manera eficiente la labor a los desarrolladores. La librería Three.js [19] surge con el objetivo de simplificar el acceso a las funciones de WebGL y WebVR. Se trata de una librería escrita en JavaScript, sirve para crear gráficos 3D animados en navegadores web y puede utilizarse sobre WebGL.

2.3.3. React VR

En abril de 2017 Facebook lanzó React VR [20]. Este nuevo *framework* de código abierto buscaba promover la creación de contenido inmersivo en la Web y facilitar a los desarrolladores el añadido contenido virtual a sus aplicaciones. Durante el año siguiente a su lanzamiento, la comunidad React VR ha utilizado los recursos de este *framework* para crear experiencias como paseos en 360 o la promoción de marcas, películas y productos a través de ambientes de inmersión. React VR sigue el diseño de React, con una estructura basada en componentes. El componente principal se carga desde el fichero *index.vr.js*, donde también se registra la aplicación para que el empaquetador la integre en la página principal (*index.html*). Igual que React y React Native, las aplicaciones cuentan con un estado y unas *props* que descienden desde los componentes padre hacia los hijos (flujo de datos unidireccional) para que estos se encarguen del renderizado. En el momento del lanzamiento existían dos versiones diferentes de React VR: la versión de código abierto centrada en el desarrollo para la web y la versión interna de Oculus [21] para sus aplicaciones móviles y para ordenadores (PC). Oculus VR es una empresa pionera en el desarrollo de tecnología de

realidad virtual, que fue comprada por Facebook en 2014. Es la responsable de dos de los visores más conocidos del mercado: Oculus Rift y Oculus Go [22]. Aunque los proyectos de ambas versiones comenzaron a partir de una API compartida, desde entonces han divergido para abordar diferentes necesidades.

2.3.4. React 360

React 360 [23] fue presentada durante la conferencia anual de Facebook, que se celebró a principios de mayo de este año. Esta versión, aunque está basada en los mismos principios y tiene los mismos objetivos que su predecesora (React VR), se centra en la mejora de aspectos relacionados con los contenidos bidimensionales (2D). Uno de los motivos principales de esta decisión ha sido que son este tipo de contenidos en los que se centra la mayoría del trabajo de los desarrolladores que quieren utilizar el *framework* de realidad virtual para enriquecer sus sitios web. De esta manera, quedan en la versión privada las herramientas centradas en los gráficos 3D.

React 360 ha sido pensada para llegar a todo tipo de usuarios, incluso para aquellos que no poseen dispositivos con los que consumir el contenido de realidad virtual de forma plenamente inmersiva. El objetivo es ofrecer una mejor experiencia de usuario en la web, pudiendo consumir de una manera más interactiva y en un entorno 360 los contenidos. En su documentación oficial se destacan las principales mejoras de React 360 respecto a React VR:

- Mayor facilidad para la creación de interfaces 2D en el espacio 3D. La introducción del concepto de Superficies (*Surfaces*) facilita la colocación de paneles tanto cilíndricos como planos en la aplicación. En ellas se puede trabajar con píxeles en lugar de en metros.
- Mejora en el soporte de medios. Existen nuevas características de entorno para mejorar los medios de inmersión. El desarrollador tiene un control más preciso sobre la apariencia de la aplicación en todo momento.
- Mejor rendimiento. Se ha reconstruido completamente el tiempo de ejecución. El objetivo de la arquitectura actual es permitir a los dispositivos de gama más baja poder disfrutar también de los contenidos RV.

Una aplicación React 360 está compuesta por el código de la aplicación React

y el tiempo de ejecución (*runtime*), código que transforma los componentes de React en elementos 3D sobre la pantalla. Los navegadores web tienen un único hilo de ejecución (*single-threaded*), por lo que si se bloquea la aplicación, el renderizado se detendría. En aplicaciones cuyo consumo se espera que sea sobre visores RV, este tipo de bloqueos podría romper la sensación de inmersión y por eso en React 360 el código de la aplicación se ejecuta en un contexto separado (mediante un *Web Worker Executor* en los navegadores actuales, y un *Iframe Executor* en entornos más antiguos). Cuando el código React crea nuevos elementos, se lo indica al tiempo de ejecución, igual que cuando el usuario genera alguna entrada con su interacción llega a React en forma de evento. Los componentes de React 360 utilizan estados y *props* a modo de datos de entrada para que los componentes hijos se rendericen según el estado del padre. El empaquetado en proyectos React 360, como con React Native, se hace con el uso de Metro, un empaquetador JavaScript que funciona de forma muy similar a Webpack. El hecho de que se trate de un *framework* tan reciente afecta tanto a la cantidad de características ya implementadas de las que se puede disponer, como a los ejemplos y desarrollos de soluciones encontradas por la comunidad de los que se tiene referencia. En React VR por ejemplo se cuenta con varios componentes, y en React Native existen actualmente más de 50 componentes y diversas API. Sin embargo React 360 cuenta con pocos componentes documentados en su página oficial, entre los que podemos encontrar varios heredados de React Native.

2.4. Comunicación entre contextos del navegador

Cuando se escribe código JavaScript para una página web, normalmente es necesario usar diferentes API Web para acceder a objetos como *Body*, *BufferSource*, *CanvasPatterns*, *DOMConfiguration*, *DOMObject*, o *EventListener* entre otros. Uno de los más comunes es *window* [24], que dispone de métodos, propiedades y eventos, y entre ellos se encuentra el método *postMessage()* [25], el cual proporciona un medio seguro para que una ventana envíe datos a otra que no necesariamente esté en el mismo dominio, ya que normalmente los *scripts* de diferentes páginas sólo pueden acceder entre ellos si las páginas comparten el mismo protocolo, número de puerto y máquina anfitriona (*host*).

Con *window.postMessage()* se puede resolver de manera segura esa restricción. Para ello, se ha de obtener una referencia de la ventana a la que se quiere enviar un mensaje (suele ser un *pop-up* o un *iframe*) y luego utilizar el método para enviar el

mensaje en un objeto de evento que el receptor será libre de manejar. *targetWindow.postMessage(message, targetOrigin, [transfer]);*

- *targetWindow*: es la referencia a la ventana receptora.
- *message*: son los datos que se van a enviar. Se serializa usando el algoritmo de clonación estructurada, por lo que se pueden enviar numerosos tipos de objetos sin necesidad de que el desarrollador deba serializarlos previamente.
- *targetOrigin*: especifica el origen de la ventana receptora para que pueda lanzarse el evento (esquema, *hostname* y puerto que se indiquen aquí deben coincidir con los del destino). Puede ser un literal *string* *, que permitiría cualquier origen, o ser un Identificador Uniforme de Recursos (URI).
- *transfer*: es opcional y se utiliza para enviar objetos transferibles al destino.

Capítulo 3

Contexto

3.1. La herramienta de autor Ediphy

Ediphy [26] [27] es una herramienta web de código libre y autoría de recursos para *e-Learning* que permite crear material educativo multimedia con una gran variedad de contenido. Esta herramienta se desarrolla en el Grupo Internet de Nueva Generación del Departamento de Ingeniería de Sistemas Telemáticos de la ETSIT-UPM. Al tratarse de una plataforma web, no es necesario instalar ningún software adicional para utilizarla y se puede consumir desde cualquier dispositivo con un navegador instalado. Es un proyecto desarrollado en Javascript (ES6), junto con librerías como React y Redux. Ediphy tiene una arquitectura modular basada en componentes, llamados *plugins*, que le permiten ser extensible y facilitar el incremento su funcionalidad de modo sencillo para los programadores. La herramienta se puede utilizar en modo editor, donde se crean los cursos, o en modo visor, donde se consumen.

3.2. Plugins y API de creación de Plugins

Tal y como se indica en la documentación oficial de la herramienta, la unidad mínima de contenido en la que se basa Ediphy es el *plugin*, concepto que alude a la representación de una caja editable y/o configurable con algún tipo de contenido esencial, como podría ser un texto. Existen diversos *plugins* que atienden a las necesidades que se originan en la creación de un recurso educativo, y en la interfaz de la herramienta aparecen agrupados en categorías según su naturaleza. Los *plugins* se pueden añadir tanto a documentos como a diapositivas (los dos posibles tipos de vista para los cursos que se crean con esta herramienta) y algunos tienen extendida su funcionalidad (enriquecidos) y permiten la creación de puntos calientes (marcas) para navegar entre vistas y a contenido externo, generar un contenido vinculado al *plugin*, o lanzar texto explicativos a modo *pop-up*. Actualmente Ediphy cuenta con numerosos

plugins que ayudan a los usuarios a disponer de una variedad amplia de funcionalidades para la creación de recursos educativos. Entre ellos podemos destacar las imágenes, gráficos, reproductores de vídeo o webs incrustadas.

Debido a la filosofía de código libre (*open source*) que sigue el proyecto, es posible contribuir creando un *plugin* propio, en el que se pueden reutilizar componentes de React ya existentes. Para ello, la herramienta cuenta con una API para la creación de *plugins* que permite a los desarrolladores partir de una base ajustada a los mínimos requisitos requeridos para la implementación de nuevas funcionalidades. Con la ejecución del comando *yarn run créate-plugin Nombre del plugin* se crea una nueva carpeta dentro del directorio de *plugins* que incluye todos los ficheros necesarios para empezar a desarrollar. La API está también preparada para especificar de entrada la categoría a la que pertenece el *plugin*, si es enriquecido o si necesita plantilla de visualización.

```
export function <NombreDelPlugin>(base){return{
  init: function(){},
  getConfig: function(state){},
  getToolbar: function(){},
  getInitialState: function(){},
  getRenderTemplate: function(state, props){},
  getConfigTemplate: function(state){},
  afterRender: function(element, state){},
  handleToolbar: function(name, value){},
  _funciones auxiliares_: function(event, element, parent){}}}
```

En el esqueleto del *plugin* para el editor que se muestra en la figura anterior, se puede ver el objeto base que recibe como parámetro. Este hace referencia al *plugin* base del que dependen todos los *plugins* de la aplicación. Es necesario para utilizar los métodos de acceso y modificación del estado, así como para registrar funciones auxiliares para que puedan ser utilizadas por cualquier otro. Las funciones más relevantes, por ser obligatorias para todos los *plugins* independientemente de la complejidad o requerimientos de su desarrollo, son *getConfig()* y *getRenderTemplate(state)*. La primera devuelve un único objeto con la configuración del *plugin*. Algunos datos configurables son el nombre, la categoría a la que pertenece, el icono que aparecerá junto al nombre, el modo en que se ha escrito (JavaScript puro o React) o los valores iniciales del aspecto de la caja en la que se presenta el *plugin*. La segunda función obligatoria tiene como objetivo devolver una cadena de texto, ya sea mediante de elementos HTML o componentes de React pero siempre devolviendo un único elemento raíz que no sea una etiqueta con autocierre.

Capítulo 4

Desarrollo

4.1. Captura de requisitos

Previo a la selección de tecnologías e implementación, se hace necesario capturar los requisitos que marcarán las líneas del proyecto. En este caso, dado que la herramienta elegida permite la creación de contenido educativo, se precisa la distinción de dos tipos de requisitos: educativos y funcionales.

4.1.1. Requisitos educativos

Dado que Ediphy es una herramienta de creación de recursos educativos, y para determinar las características del plugin que se pretende añadir, se han tenido en cuenta los requisitos educativos a los que se debe ajustar.

- Proporcionar entornos de realidad virtual que permitan al usuario con rol de alumno aprender de una forma diferente y simulen su inmersión en ambientes de los que puede extraer un aprendizaje
- Añadir la posibilidad de fragmentar la información en pequeñas píldoras de contenido
- Facilitar la relación de la información proporcionada y el entorno virtual educativo para poder ajustarlos fácilmente al fin educativo

Un caso de uso podría ser aprender los planetas y sus características más importantes. A través de un paseo virtual por el espacio, se podrá hacer clic sobre cualquier planeta o astro y obtener su información. Es una forma de ofrecer al usuario con rol de alumno una forma de aprendizaje lúdica, basada en la observación de un entorno virtualizado que simula el desplazamiento a un planetario sin movernos del aula.

4.1.2. Requisitos funcionales

Al desarrollar contenidos educativos enriquecidos con el uso de tecnologías de realidad virtual se busca proveer al usuario de un entorno más dinámico e inmersivo.

- Uno de los requisitos principales que debe tener una aplicación de estas características es la posibilidad de añadir componentes capaces de proporcionar contextos, y que permitan generar escenarios con profundidad y realismo. Además, en este sentido será necesario que el usuario pueda elegir sus propios escenarios.

- La aplicación también debe permitir que se añada audio de fondo para aumentar el grado de inmersión en los escenarios de 360. El sonido ambiente es un elemento que contribuye a la sensación de inmersión, ya que permite introducir al usuario dentro del mundo virtual no sólo a través del sentido de la vista, sino utilizando la capacidad auditiva.

- Otro requisito funcional será la inclusión de objetos con los que interactuar, que permitan ser colocados a diferentes distancias para nuevamente aumentar la sensación de inmersión, y faciliten el aprendizaje mediante el consumo de información en pequeñas píldoras, en principio textuales.

- Por último, se estima necesario disponer de un elemento a modo carrusel de imágenes que permita completar de manera más gráfica la información dada en el recurso.

En resumen, estos son los requisitos identificados que se traducirán en funcionalidades y permitirán construir paseos tridimensionales en entornos virtuales:

- Escenarios 360 con fondo personalizados
- Activar/desactivar Audio ambiente
- Objetos tridimensionales
- Botones con los que interactuar
- Proyección de imágenes 2D

4.2. Selección de tecnologías

Ediphy es una herramienta compleja, con numerosas funcionalidades y un desarrollo en continua evolución. Utiliza React como base para toda su arquitectura y por este motivo React VR, una librería del mismo grupo para la creación de contenido de realidad virtual, se consideró una buena opción. Esta librería tenía pocos meses de vida al comenzar este trabajo (abril del 2017), pero en el contexto de Ediphy se presentaba como la mejor opción para ampliar las funcionalidades de la herramienta con el desarrollo de un *plugin* de realidad virtual. React VR está basada en React Native, y ambas librerías comparten los módulos y las API básicos. Sin embargo, React VR tiene características diferentes que la dotan de posibilidades para crear contenido de realidad virtual en la web mediante componentes específicos.

Durante el desarrollo del trabajo, concretamente a comienzos de mayo de este año, Facebook decidió abandonar la línea de trabajo que había seguido su librería de realidad virtual y renombrarla en una nueva versión que reconduce hacia objetivo de permitir a los desarrolladores crear contenido de realidad virtual para consumir en los navegadores web sin necesidad de conocer conceptos nativos de este tipo de tecnologías, sino basándose en el uso de JavaScript, para crear así contenido más inmersivo, interactivo y atractivo para los usuarios. Esta nueva librería se conoce como React 360 y cambia el funcionamiento de los componentes principales y las características del flujo de ejecución, aunque mantiene algunas similitudes con su predecesora. Sobre esta librería es sobre la que finalmente se ha implementado la aplicación que se añade a Ediphy para ampliar sus funcionalidades, poniéndola a la vanguardia de las aplicaciones de autoría de recursos educativos que disponen de herramientas de realidad virtual.

4.2.1. Investigación React VR

React VR se centra en aspectos típicos de las aplicaciones 3D, es decir, priman los objetos tridimensionales pensados para ser renderizados en el espacio, según un sistema de coordenadas basado en metros. Al ser una librería React, la aplicación se construye mediante una arquitectura basada en componentes, que tienen un único método obligatorio: *render()*. Desde este método se renderizan en la pantalla los componentes necesarios para construir una escena o parte de ella. Esta es una lista de algunos de los componentes con los que cuenta React VR:

- *Pano*: controla el fondo de la aplicación. Recibe imágenes en su atributo *source* y las renderiza en un entorno 360.
- *VideoPano*: similar a Pano, pero en este caso utiliza como fuente vídeos grabados por cámaras especiales, en 360.
- *Model*: renderiza objetos creados en 3D. Admite también entre sus atributos el archivo de materiales (*.mtl*) correspondiente si es necesario.
- *VrButton*: se encarga de envolver a otros componentes para dotarlos de la interactividad típica de los botones, como son los eventos de *click*, focalización sobre ellos o selección. No tienen un aspecto físico por sí solos, sino que deben servir como envoltorio de otros componentes como <*Text*>, <*Image*> o <*Model*>.
- *Image*: renderiza imágenes 2D en posiciones respecto del eje de coordenadas del mundo virtual, es decir, en puntos con dimensiones.
- *Video*: renderiza vídeos 2D en posiciones respecto del eje de coordenadas del mundo virtual, es decir, en puntos con dimensiones.
- *Sound*: habilita una fuente de sonido que emite el contenido del archivo de audio que se pase al atributo de fuente en el componente en el que está envuelto.
- *View*: al igual que <*VrButton*>, este componente no tiene un aspecto físico propio que renderizar, sino que sirve para envolver a otros componentes. Se puede encontrar una equivalencia en HTML con la etiqueta <*div*>.
- *Text*: renderiza texto en 2D en posiciones respecto del eje de coordenadas del mundo virtual, es decir, en puntos con tres dimensiones.

En React VR, tanto la renderización de elementos 3D en la pantalla del navegador como la lógica de la aplicación se ejecutan en un único subproceso (los navegadores tienen un único hilo de ejecución principal). Este hecho provoca que cualquier retardo que cause el código de la aplicación, hará que aumente la latencia en el renderizado de la escena tridimensional. Esto es un aspecto crítico a la hora de consumir los contenidos de realidad virtual en visores, ya que se pierde la sensación de inmersión. En React VR toda la aplicación se escribe a partir del fichero *index.vr.js*, donde se registra el componente principal y *client.js* se encarga de prepararlo para su compilación e integración como JavaScript en *index.html*.

4.2.2. Primera aproximación: React VR

Durante la primera parte de la elaboración de este trabajo, la librería vigente era React VR y la propuesta de escenario se centraba en el uso de objetos 3D.

El profesor selecciona un fondo para ambientar su plugin de realidad virtual en Ediphy. Para ello puede elegir entre proporcionar el Localizador Uniforme de Recursos (URL, por sus siglas en inglés) de un recurso de tipo imagen 360 o vídeo 360, o seleccionar alguno de los que ya dispone la aplicación por defecto. Introduce, mediante la carga de archivos que contengan objetos en 3D, modelos en el espacio tridimensional, en las coordenadas y con la escala que elija para cada uno. Puede elegir si animarlos de manera que roten sobre su propio eje vertical o no, dando así la posibilidad de ofrecer más dinamismo a la escena. El profesor, durante la edición del componente, puede introducir también recursos tales como vídeos o imágenes, para que sean proyectados a una distancia determinada del usuario, además de habilitar los controles básicos de los mismos (reproducir, parar, siguiente y anterior). El alumno consume el contenido generado por el profesor de manera que puede navegar por el mundo de realidad virtual, observando los modelos o recursos gráficos que se hayan añadido, y controlando estos últimos en el caso de que se hayan habilitado los controles para ello.

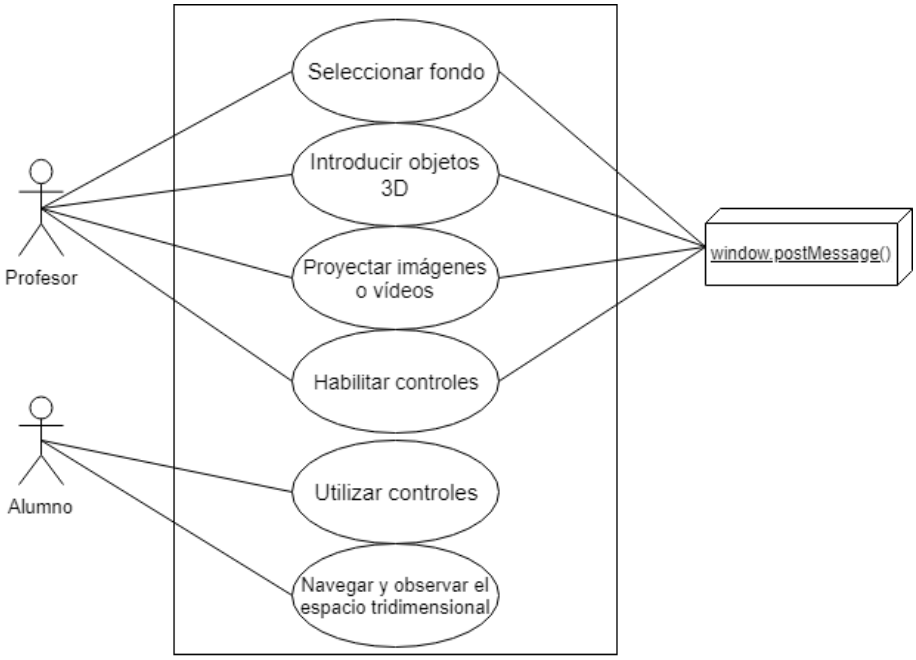

Figura 4 : Escenario propuesto con React VR

Se hicieron dos pruebas de concepto utilizando los elementos que se describen en el escenario para comprobar la idoneidad del resultado antes de empezar a implementar una aplicación que permitiese la edición desde Ediphy. La primera se basó en varios ejemplos que los desarrolladores han ido publicando durante el año que ha estado lanzada esta librería. Se probaron botones animados en forma de texto y de imágenes, sus funciones para cambiar la imagen de fondo y un menú para elegir entre determinados vídeos que fueran proyectados.

Figura 5 : Prueba de Concepto con React VR

El componente principal de la aplicación, el que renderiza *index.vr.js*, tiene un estado desde el que se controlan el tipo de cambio de fondo (manual o automático) y el vídeo que se proyecta. El resto de los componentes se han construido basándose en los más básicos ofrecidos por la librería, como son <*View*> para agrupar, <*VrButton*> para dotar a las imágenes y textos de los menús de animación e interactividad, <*Pano*> y <*VideoPano*> para la imagen o vídeo de fondo, <*Sound*> para el sonido ambiente, <*Image*> y <*Video*> para los recursos audiovisuales que se proyectan y <*Text*> para el contenido de los elementos del menú. La segunda prueba consistió en replicar uno de los ejemplos más conocidos de está librería en internet [28]. Se trata de pruebas de concepto que finalmente no llegaron a utilizarse en la aplicación final de realidad virtual para Ediphy por el cambio de librería.

4.2.3. Investigación React 360

React 360 surge para romper con la tendencia que marcaba el enfoque tridimensional de React VR, y se centra en la mejora de la renderización de contenido en 2D dentro de un mundo virtual 360, adecuándose a la línea de trabajo que persiguen desde Facebook con esta versión de código abierto, que se basa en la animación y mejora de la interactividad de contenidos web. Una de las mayores diferencias entre las dos

librerías es la utilización de que lo que se denomina tiempo de ejecución. Se trata de separar el contexto de la aplicación del renderizado, es decir, de la conversión de código en React a elementos 3D en la pantalla. De esta forma, se permite al ciclo de representación que se actualice constantemente a una velocidad de fotogramas alta y cuando el código React crea nuevos elementos, le indica al tiempo de ejecución que los agregue a la escena 3D. En sentido contrario, si el usuario proporciona una entrada, se devuelve a React en forma de evento. En este caso, el fichero en el que se aloja el código principal de la aplicación es *index.js*, en el cuál se importan el resto de los componentes que cuelgan de este para formar parte de la aplicación. El tiempo de ejecución, es el código del fichero *client.js*, el cual conecta la aplicación con el navegador. En este fichero se crea una nueva instancia de React 360, se carga el código React y se conecta a un lugar específico en el DOM (proporcionado por *index.html*), además de tener las opciones de inicialización que especifique el desarrollador. Posteriormente, este archivo se encarga de renderizar el código React en la escena, adjuntando el componente declarado en *index.js* a la superficie predeterminada y el resto de componentes (si los hubiera) a las superficies o localizaciones que se indique. Opcionalmente, se puede agregar una imagen de fondo de 360 para mostrar mientras se carga el código React, mejorando la percepción del usuario.

La otra diferencia importante en la que React 360 mejora las características de React VR es en la inclusión del módulo Superficie (*Surface*). Con él, se facilita la creación de interfaces 2D en el espacio 3D, pudiendo trabajar en píxeles en lugar de en metros. De esta manera, se pueden utilizar especificaciones creadas con herramientas de diseño tradicionales y también secciones de código de React Native. Algunos de los componentes con los que cuenta son heredados de React VR (*View*, *Text*, *Image*, *VrButton*). Sin embargo, *Model* ha pasado a llamarse *Entity* y admite objetos por ahora en formatos *.obj* y *.gltf*, con su correspondiente fichero de materiales *(.mtl)*. Además, React 360 cuenta con varias API:

- *Animated*: fue implementada originalmente para React Native y actualmente es esta misma la que se usa para la librería de realidad virtual de React. Con ella se consiguen alternar el tamaño, la posición y la apariencia de los elementos en la escena de forma que se generan efectos que ayudan y enriquecen la interactividad del usuario con dichos elementos.

- *AsyncStorage*: se trata de una API JavaScript simple de almacenamiento de clave-valor que permite conservar datos entre múltiples cargas de una aplicación en el navegador.

- *ControllerInfo*: proporciona información sobre dispositivos externos tales como mandos y controladores conectados al ordenador. Se utiliza para responder a los eventos de conexión y desconexión del controlador y se puede extraer información estática sobre los controladores (identificadores únicos, recuentos de botones y ejes, izquierda a derecha, etc.).
- *Environment*: se utiliza para cambiar el aspecto del fondo de la escena, ya sea mediante el uso de imágenes estáticas con formatos que permitan su renderización en un entorno 360 (mono/estéreo equirectangular), o mediante el uso de vídeos en estos formatos. Sustituye la funcionalidad que tenía *Pano* y *VideoPano* en ReactVR.

Utilizando los ejemplos iniciales con los que se cargan los proyectos de estas librerías, se puede apreciar la diferencia de enfoques que se ha comentado sobre el uso que desde Facebook pretenden que los desarrolladores hagan de su librería.

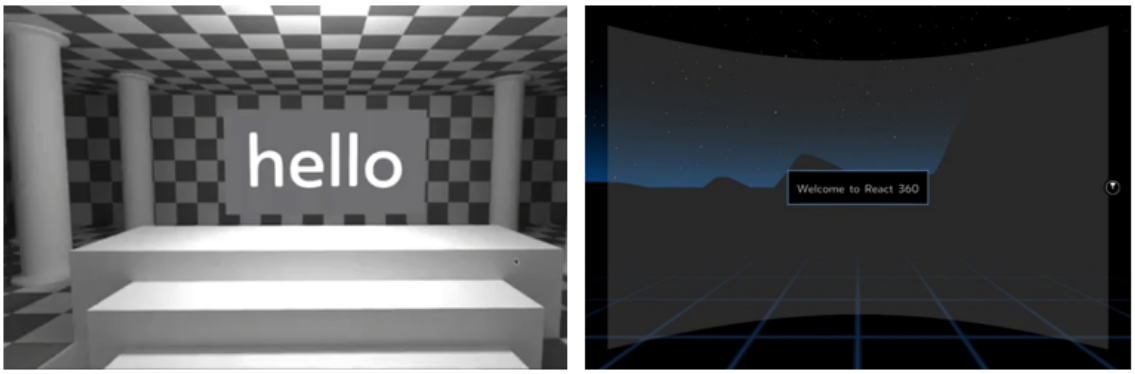

Figura 6 : Comparativa de proyectos iniciales

4.2.4. El escenario definitivo: React 360

Tras el cambio de librería, la propuesta de este trabajo tuvo que reevaluarse en función de las nuevas características que se ofrecen en React 360. En la página web oficial se indica que, para facilitar la transición de una a otra, los componentes de React VR aún están disponibles en el nuevo paquete. Sin embargo, esto puede cambiar en el futuro, ya que se ha reestructurado el funcionamiento y la manera en la que se renderizan algunos componentes, así que puede que terminen por quedar obsoletos.

El profesor puede seleccionar un fondo para ambientar su plugin de realidad virtual en Ediphy. Para ello puede elegir entre proporcionar la URL de una imagen 360,

o seleccionar alguna de las que ya dispone la aplicación por defecto. Introduce, mediante herramienta para plugins enriquecidos de Ediphy, marcas en el espacio tridimensional, en las coordenadas que elija para cada una. Estas marcas pueden ser de tipo emergente (muestran un texto al ser seleccionadas) o enlazar con otra diapositiva o página del curso que está editando. Puede elegir si habilitar los controles básicos del audio ambiente (audio por defecto). El profesor puede seleccionar también hasta diez imágenes, para que sean proyectados a una distancia determinada del usuario. El alumno consume el contenido generado por el profesor de manera que puede navegar por el mundo de realidad virtual, observando los puntos marcados y las imágenes del proyector, así como controlar la reproducción o pausa del audio ambiente, en caso de disponer de los controles para ello.

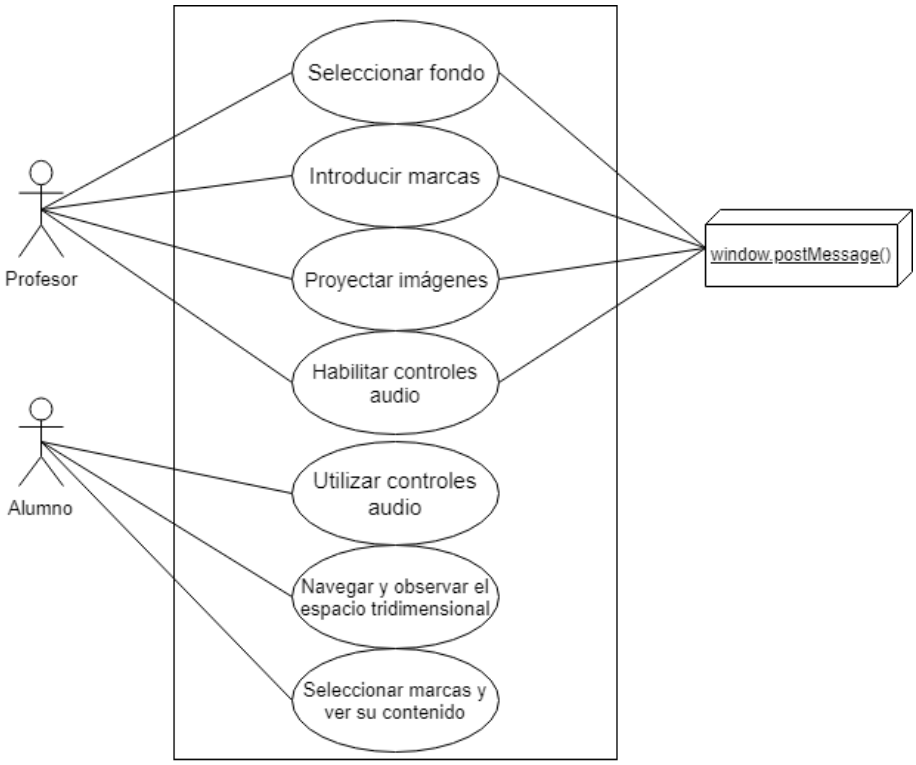

Figura 7 : Escenario propuesto con React 360

Para este escenario no se realizó ninguna prueba de concepto, ya que el hecho de que cambiase la librería base del trabajo en mitad del desarrollo provocó la necesidad de redefinir el objetivo del *plugin*, estudiando de nuevo la documentación y las posibilidades de la nueva librería, de manera que el desarrollo debía ser lo más eficiente posible en términos de tiempo dedicado a cada paso para implementar la aplicación final.

4.3. Implementación

4.3.1. Integración de React 360 en Ediphy

Según las indicaciones de la documentación oficial, existen dos maneras de integrar la aplicación de realidad virtual en un sitio web ya existente: mediante el uso de *iframe* o añadiendo el tiempo de ejecución a una aplicación JavaScript mayor. Sin embargo, la segunda opción no se recomienda debido a la necesidad del código de la aplicación de ser empaquetado previamente, lo cual requiere añadir ciertas funcionalidades extras al empaquetador que utilice la aplicación en un principio, además de que supone una carga extra de varios cientos de Mega Bytes. Por lo tanto, para integrar esta aplicación en Ediphy se decidió seguir la recomendación oficial y utilizar la vía del *iframe*. Mediante el atributo (*src*) de la etiqueta HTML <*iframe*>, se apunta al archivo *index.html* de la aplicación de realidad virtual. De esta manera, no se interfiere con la página web principal y así esta no ralentiza la renderización de la aplicación.

Al igual que todas las librerías de React, durante el desarrollo de la aplicación el código se compila y empaqueta dinámicamente mediante un servidor local que lanza la aplicación y queda disponible en un puerto de la máquina (concretamente, por defecto es el 8081). Por otro lado, cuando la implementación del código de la aplicación ha terminado, el proyecto se pasa a producción ejecutando uno de los *scripts* que tiene por defecto React 360. En producción, todo el código necesario para ejecutar la aplicación es empaquetado en un solo directorio, que contiene la compilación de producción del código JavaScript y una copia del archivo *index.html* que apunta a la compilación JavaScript. Si en la aplicación se utilizan recursos alojados en la carpeta de archivos estáticos (*static_assets*), esta debe copiarse también dentro del directorio de producción, en el servidor en el que vaya a alojarse la aplicación. En el caso de este proyecto, una vez terminado el desarrollo, el directorio de producción se lleva a la carpeta *dist* en Ediphy y se modifica la dirección absoluta a la que apuntaba el atributo *src* del *iframe*, por una ruta relativa al fichero *index.html* de producción. Una vez determinada la forma de integrar la aplicación React 360 en Ediphy, el siguiente objetivo fue comunicarlas. La pregunta era: £cómo hacer que dos ventanas diferentes (un *iframe* incrustado en una página web) puedan enviarse mensajes entre sí? El propio elemento de las ventanas en los navegadores (*window*) posee un método cuya descripción indica que su propósito es precisamente permitir de forma segura la comunicación entre ventanas con distinto origen, ya se trate de páginas web independientes, un *iframe* dentro de una página o una

ventana emergente. Es el método *postMessage()* (la explicación de este método puede verse detallada en el Estado del Arte). Mediante el uso de esta función, se ha diseñado un protocolo de mensajes para la conexión y envío de información entre el *iframe* que carga la aplicación React 360 y la web de Ediphy. Esta información contiene los datos correspondientes al estado de la aplicación y con ellos los componentes de la misma se renderizan como corresponde. En el siguiente diagrama puede verse un ejemplo de interacción del usuario con la aplicación Ediphy, utilizando el *plugin* de realidad virtual para poner una marca en cierto lugar al que ha rotado moviendo la cámara. Luego, en previsualización, seleccionará la marca.

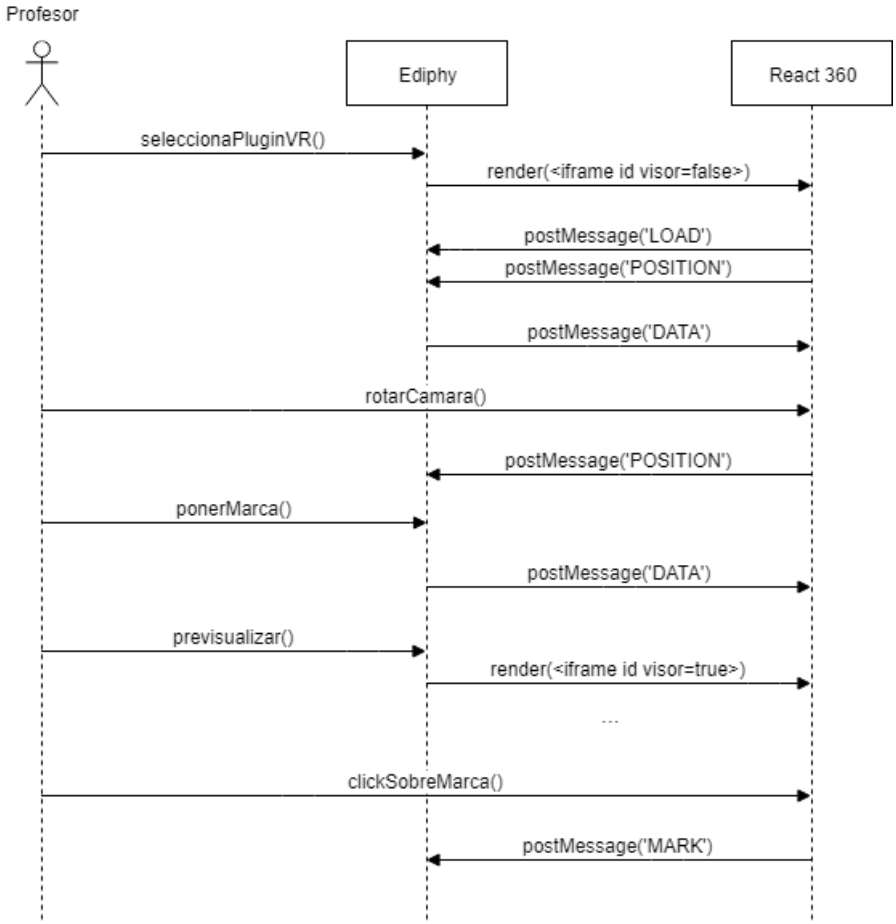

Figura 8 : Protocolo de mensajes con *postMessage()*

Como puede observarse en la figura, Ediphy reacciona a los mensajes que recibe en función de los parámetros. La conexión entre las dos aplicaciones la empieza la de realidad virtual, con el mensaje LOAD. Esta decisión se tomó tras hacer pruebas enviando mensajes desde Ediphy con los datos del estado. Al intentar acceder a la

ventana de previsualización (la cual renderiza un nuevo *iframe*), la aplicación no se cargaba porque no recibía el estado. El resultado era que en el visor no se conseguían visualizar los cambios aplicados en el editor, sino que la aplicación 360 se quedaba en el estado inicial. Este fallo quedó solventado con el nuevo orden en el protocolo, cuyos mensajes indican la carga completa de la aplicación para que Ediphy pueda enviar eficazmente los datos (mensaje DATA) sobre el estado del *plugin* editado. Para enviarlos, se tuvo que comprobar que el atributo identificador (*id*) de la ventana que envía el mensaje LOAD se corresponde con el *id* que tiene guardado Ediphy para ese *iframe* concreto.

```
let query = decodeURIComponent(window.location.search);
try{
  let myurl = query.split("?id=");
  var sinNomId = myurl[1];
  var sinNomVisor = sinNomId.split("&visor=");
  var id = sinNomVisor[0];
  idBox = id;
  window.parent.postMessage(JSON.stringify({msg: 'LOAD', id: id}),"*")
```

```
let data = JSON.parse(e.data);
if (!this.windowSource && data.msg === 'LOAD' && data.id === this.props.id) {
    this.windowSource = e.source;
    this.toolbarUpdateValue();}
```

```
toolbarUpdateValue(props = this.props) {
    let receiverWindow = this.windowSource;
    if(receiverWindow) {
        let { imagenBack, urlBack, audioBack, showPanel, numberOfPictures } =
            props.state;
        let imgs = [];
        for (let i = 0; i < numberOfPictures; i++) {
            imgs.push({ currentImg: props.state['urlPanel' + i] });}
        receiverWindow.postMessage({ msg: 'DATA', imagenBack, urlBack, audioBack:
             { play: audioBack },
            showPanel: { show: showPanel }, imgs, marks: props.marks }, "*");
}}
```

La aplicación al cargarse envía también un mensaje POSITION con las coordenadas de la posición actual a la que se está mirando (rotación). Este dato se comprueba cada 2 segundos y se compara con el último calculado de manera que, si ha cambiado la posición a la que mira el usuario, se envía un nuevo mensaje POSITION a Ediphy.

```
setInterval(()=>{
     let currRot = VrHeadModel.rotation();
     if(JSON.stringify(this.state.currRot) === '[]')      this.setState({ currRot:
         currRot });
     if(JSON.stringify(currRot) !== JSON.stringify(this.state.currRot)){
       this.setState({ currRot: currRot });
       ConexionModule.handlePosition(currRot);
     }},2000)
```

```
handlePosition(position){
       window.parent.postMessage(JSON.stringify({msg:"POSITION", id: idBox, position
           }), "*");}
```

```
if (this.windowSource && data.msg === 'POSITION' && data.id === this.props.id) {this
    .setState({ position: data.position });}
```

Por otro lado, existe otro tipo de mensaje para indicar desde la aplicación React 360 que se ha pulsado sobre una marca (mensaje MARK) cuya funcionalidad permite navegar por el recurso de Ediphy o ir a un contenido externo. Para ello, se le indica a la herramienta el *iframe* en el que está la marca y sus datos.

```
   handleMark(mark, id){
       window.parent.postMessage(JSON.stringify({msg:"MARK", id, mark}), "*");}
```

```
  if (this.windowSource && data.msg === 'MARK' && data.id === this.props.id) {
       this.props.onMarkClicked(this.props.id, this.props.marks[data.mark].value);}
```

Este segundo parámetro de la función postmessage indica cuál debe ser el origen de la ventana de destino para que el evento se envíe. Si el esquema, el nombre de la máquina o el número de puerto no coinciden con los del documento de la ventana de destino, el evento no se envía. En los fragmentos de código anteriores puede verse cómo ese parámetro se envía con valor *. De esta forma no se comprueba el destino al que se envían los datos.

4.3.2. Aplicación React 360: Módulo de conexión

Una de las diferencias fundamentales entre React VR y React 360 es el uso del tiempo de ejecución para mejorar el rendimiento de la aplicación. Esta separación de

contextos hace que desde la aplicación de React no se pueda acceder a ciertos métodos de la API web (como son los que necesita nuestro proyecto: *window.postMessage()* y *window.addEventListener()*) que el navegador tiene disponible sólo para el hilo principal. La solución que ofrecen los desarrolladores de la librería es usar módulos nativos. Algunos de los módulos más comunes ya están disponibles en el paquete de React 360 para web, como *Location* y *History*.

Los módulos nativos extienden la clase *Module* y deben crearse y registrarse durante el inicio de la aplicación (en *client.js*). Deben ser importados en los componentes que vayan a usarlos.

```
import ConexionModule from './ConexionModule';
function init(bundle, parent, options = {}) {
 const r360 = new ReactInstance(bundle, parent, {
   fullScreen: true,
   nativeModules: [ctx => new ConexionModule(ctx),],...options,
 });
```

Para la aplicación que se desarrolla en este trabajo se ha implementado un módulo nativo llamado *ConexionModule*, cuyo objetivo es recibir y enviar mensajes utilizando *postMessage()* y determinar, según los parámetros con los que se llama a la aplicación, cuál es su identificador en Ediphy y si se encuentra o no mostrándose en el visor. El código de este elemento comienza importando la clase *Module* del paquete *react-360-web*, ya que el modulo nativo que se cree debe extender esta clase. En el constructor se llama al primero al de la clase a la que extiende pasándole como parámetro el nombre del modulo nativo que se crea, de forma que quede accesible en el componente *NativeModules* de React 360 en *NativeModules.ConexionModule*. Se inicializa el contexto y los datos de ventana y origen que se utilizarán para conectarse con la ventana en la que se inserte la aplicación vía *iframe*.

```
  import {Module} from 'react-360-web';
  let idBox = null;
  export default class ConexionModule extends Module {
    constructor(ctx) {
      super('ConexionModule'); // Makes this module available at NativeModules.
          MyModule
      this._rnctx = ctx;
      this.winSource = undefined;
      this.origin = undefined;
      this.handleMark = this.handleMark.bind(this);
    }
```

A continuación, se detallarán las funciones que implementa este módulo, las cuales quedan disponibles para ser usadas por el código de la aplicación al ser importado desde *NativeModules* (y previamente registrado en el *client.js*).

- *conexionIframe(callback)*: Esta función tiene dos partes y se llama antes de renderizar los componentes principales, los cuales necesitan que Ediphy les envíe cierta información para poder pintarse correctamente. La primera parte no depende de ningún evento, sino que directamente recoge de la *query* el parámetro de identificación y se lo envía en un mensaje LOAD a la ventana en la que se encuentra incrustada la aplicación. En la segunda, se espera por una promesa que recogerá los datos que envíe Ediphy cuando haya validado la conexión gracias un escuchador de eventos que habilita para ello. Además, guarda en las variables que se comentaron en el constructor para que queden accesibles al resto de funciones, sin necesidad de esperar a recibir ningún evento del que sacar la información del destino.

```
conexionIframe(callback) {
     const result = new Promise((resolve, reject) => {
       window.addEventListener("message",  function(event) {
         this.winSource = event.source;
         this.origin = event.origin;
         if(event.data.audioBack){
           this.winSource.postMessage(JSON.stringify({msg:"Estado audio
               recibido correctamente"}), this.origin);}
         if(event.data.urlBack){
            this.winSource.postMessage(JSON.stringify({msg:"Url Back recibida
                 correctamente"}), this.origin);}
         if(event.data.imagenBack){
           this.winSource.postMessage(JSON.stringify({msg:"Imagen Back
               recibida correctamente"}), this.origin);}
         if(event.data.imgs){
           this.winSource.postMessage(JSON.stringify({msg:"Imágenes recibidas
                correctamente"}), this.origin);}
         if(event.data.showPanel){
           this.winSource.postMessage(JSON.stringify({msg:"Show Panel
               recibido correctamente"}), this.origin);}
         if(event.data.marks){
           this.winSource.postMessage(JSON.stringify({msg:"Marks recibidas
               correctamente"}), this.origin);}
         resolve(event.data);
       }),function e(){console.log("Problemas con la promesa");};
     });
     result.then(datos => {
       if (this._rnctx) {this._rnctx.invokeCallback(cb, [datos]);}
     });
```

La fución *callback* se encarga de recoger los datos necesarios para cada componente y actualizar su estado según sea necesario. Por ejemplo, en el componente que

controla las marcas (*Mark*), se invoca esta función desde el *componentDidMount()* y con el resultado se actualiza el estado con las marcas que Ediphy tiene registradas para esta aplicación (identificada por el campo id).

```
escucharConexion() {
 ConexionModule.conexionIframe(datos => {
   if (datos.marks) {this.setState({marks: datos.marks})}
   this.escucharConexion();
 });}
```

En el proyector los datos sirven para guardar el JSON de direcciones de los recursos que se ha decidido colocar en el panel.

```
escucharConexion() {
 ConexionModule.conexionIframe(datos => {
   if(datos.imgs && datos.imgs.length > 0){
     this.setState({
         arrayImgs: datos.imgs,
         keySelected: 0,
         currentImg: datos.imgs[0].currentImg,});
   }
   if(datos.showPanel){
       this.setState({
         showPanel: datos.showPanel.show,});
   }
   this.escucharConexion();
 });}
```

Finalmente, el componente *index.js* se utiliza para la configuración del ambiente:

```
escucharConexion() {
   ConexionModule.conexionIframe(datos => {
     if(datos.audioBack){
       /*try{AsyncStorage.setItem('showAudio', datos.audioBack.play);}catch(
           error){}*/
         this.setState({
           showAudio: datos.audioBack.play
     });}
     if(datos.urlBack){
       /*try{AsyncStorage.setItem('urlBack', datos.urlBack);}catch(error){}*/
       this.setState({
         urlBack: datos.urlBack
     });}
     if(datos.imagenBack){
       /*try{AsyncStorage.setItem('imgBack', datos.imagenBack);}catch(error)
           {}*/
       this.setState({
         imgBack: datos.imagenBack
     });}this.escucharConexion();});
 }
```

Como puede verse en los comentarios del código, se ha probado a utilizar la API JavaScript *AsyncStorage*, que sirve para guardar los datos del estado en el navegador. Esta funcionalidad se dejó comentada porque no se ha necesitado durante la creación y desarrollo de la aplicación.

- *enVisor(callback)*: De la misma manera que se extrae el identificador de la caja del *plugin*, también puede extraerse el valor booleano que indica si el *iframe* en el que se está consumiendo la aplicación está en modo visor o editor.

```
enVisor (callback){
    let query = decodeURIComponent(window.location.search);
    try{
      let myurl = query.split("?id=");
      var sinNomId = myurl[1];
      var sinNomVisor = sinNomId.split("&visor=");
      var visor = sinNomVisor[1];
      if (this._rnctx) {this._rnctx.invokeCallback(cb, [visor]);}
    }catch(e){}
}
```

Esta información la utilizan tanto las marcas como la función que reproduce el audio ambiente, para deshabilitar el contenido que sólo debe consumirse desde el visor (siguiendo la política de Ediphy).

```
 handleMarkClick(){
    switch (this.props.connectMode) {
      case 'popup':
      let visor = ConexionModule.enVisor(visor => {
        if(visor === 'true'){this.setState({show: !this.state.show});}
      });return;
      default:
        this.props.onClick(this.props.id, this.props.origin)
  }}
```

En el caso de que la marca no sea de tipo emergente, sino que la acción asociada sea enlazar con otro contenido del curso, la comprobación no es necesaria ya que el componente del visor no tiene forma de interpretar los mensajes MARK ni lanzar el método pertinente (ver apartado de creación del *plugin* en Ediphy).

```
 _playAudio = () => {
  let visor = ConexionModule.enVisor(visor => {
    if(visor === 'true'){
      AudioModule.playEnvironmental({source: asset('audio/Blue_Jacket.mp3'),
           volume: 0.7,});}});
 };
_stopAudio = () => {AudioModule.stopEnvironmental();};
```

En el caso de la función que para el audio no es necesaria esta información.

- *handleMark(mark, id)*: Igual que en el apartado en el que se explica la estructura del *plugin* se indicó la función que invocaba la acción asociada a una marca que es seleccionada, en este caso lo que se tiene es la función que envía los datos necesarios desde la aplicación de realidad virtual hacia Ediphy, para que este sepa qué debe lanzar. Es decir, envía el mensaje MARK (ver apartado de comunicación).

- *handlePosition(position)*: Como la función anterior, este caso también ha sido previamente mencionado a la hora de explicar el protocolo de mensajes que intercambian la aplicación y Ediphy. Se trata del método que envía el mensaje POSITION, invocado cuando se han producido cambios en la rotación desde el último intervalo (ver apartado de comunicación).

4.3.3. Plugin Realidad Virtual en Ediphy

Se ha utilizado el comando *yarn run create-plugin VirtualReality* para la creación del *plugin* de realidad virtual que se desarrolla en este trabajo. Tal y como indica la documentación oficial, para añadirlo a Ediphy basta con ponerlo en la lista de *plugins* del fichero */core/config.es6*.

Configuración del *plugin* en modo editor(*VirtualReality.js*)

Una vez se dispone del esqueleto del *plugin*, los dos métodos obligatorios de este para el editor son *getConfig()* y *getRenderTemplate()*. Además, para editar la aplicación será necesaria una barra de herramientas y un estado incial.

- *getConfig()*: Se usó para dar nombre y categoría al *plugin*. El nombre con el que se muestra a los usuarios (propiedad *displayName*) se selecciona según su valor en el fichero de traducciones (Ediphy está disponible en inglés y español) y se agrupó con los *plugin* multimedia. El *plugin* está escrito en código React, no necesita configuración previa al lanzamiento y debe tener unas dimensiones adecuadas para entrar en la diapositiva o documento en el que se inserte. El *plugin* debe configurarse como enriquecido (*isRich*) y definir el tipo de marcas (en este caso, al ser marcas en una escena tridimensional deben tener 3 coordenadas). La propiedad *needsPointerEventAllowed* es necesaria para permitir al usuario que elija entre

mover la escena de realidad virtual o la caja que contiene el *plugin* y lo sitúa en el documento.

```
getConfig: function() {
            return {
                name: 'VirtualReality',
                displayName: i18n.t('VirtualReality.PluginName'),
                category: "multimedia",
                flavor: "react",
                needsConfigModal: false,
                needsTextEdition: false,
                initialWidth: '450px',
                initialHeight: "auto",
                initialWidthSlide: '60%',
                initialHeightSlide: '50%',
                icon: 'event_seat',
                needsPointerEventsAllowed: true,
                isRich: true,
                marksType: [{ name: i18n.t("HotspotImages.pos"), key: 'value',
                    format: '[x,y,z]', default: '0,0,0', defaultColor: '#17
                        CFC8' }],
};}
```

- *getRenderTemplate()*: Para disponer de los métodos que tienen todos los componentes de React, se creó un componente para renderizar el *plugin*. Es decir, desde el propio *plugin* no podía llamarse a métodos como *componentDidMount()* o *componentWillReceiveProps(nextProps)*, que eran necesarios para la implementación, de modo que se desagregó el componente del *plugin*. Como puede observarse en el código anterior, a este componente se le pasan como atributos (*props*) el identificador único del *plugin*, el estado y las marcas (array vacío si no se han colocado marcas). La zona para soltar las marcas del *plugin* enriquecido se encuentra configurada para los elementos con clase *dropableRichZone* en Ediphy, así que se añadió un bloque de las mismas dimensiones que el componente de realidad virtual, para dejar total libertad de inserción de marcas.

```
        getRenderTemplate: function(state, props) {
            let marks = props.marks || {};
            let id = props.id;
            return (<div style={{ height: "100%", width: "100%" }} className={
                'VRPlugin'}>
                    <div className="dropableRichZone" style={{ height: "100%",
                        width: "100%", position: 'absolute', top: 0, left: 0
                            }} />
                    <VirtualRealityPluginEditor id={props.id} state={state}
                        marks={marks} /></div>);
        }
```

- *getInitialState()*: Esta función devuelve el estado inicial de la aplicación. Tiene información relacionada con el estado de las funcionalidades de la aplicación React 360 (se hablará de ellas más adelante).

```
getInitialState: function() {
        return {
            imagenBack: undefined,
            urlBack: undefined,
            audioBack: false,
            showPanel: false,
            numberOfPictures: 1,
};}
```

  Se explicará la función de cada uno de los parámetros del estado a medida que se necesiten para las funcionalidades que se detallan más adelante.

- *getToolbar()*: Desde aquí pueden estructurarse las opciones de la barra de herramientas. Cada pestaña tiene un nombre y tantos acordeones como sean necesarios para el *plugin*. A su vez, los acordeones tienen una clave y un nombre, así como un icono relacionado con las herramientas que agrupa. Estas herramientas son botones, objetos con propiedades como nombre, tipo, opciones para los tipos de selección, icono, valor, etc. Esta función recibe como parámetro el estado del *plugin*, de manera que puede actualizarlo cada vez que el valor de alguno de sus botones cambia. Concretamente, para la aplicación de realidad virtual son necesarios dos acordeones, configuración y fondo, además del que se añade al ser un *plugin* enriquecido para poner marcas.

Componente React para el editor (*VirtualRealityPluginEditor.js)*

Del componente deben destacarse dos aspectos fundamentales: es el encargado de renderizar el *iframe* mediante el que se integra la aplicación de React 360 en Ediphy, y maneja los mensajes de conexión tanto para establecer la comunicación como para enviar el estado y recibir la posición. El código del componente se muestra a continuación.

```
constructor(props) {
    super(props);
    this.state = {position: [0, 0, 0],};
    this.toolbarUpdateValue = this.toolbarUpdateValue.bind(this);
    this.receiver = this.receiver.bind(this);
}
render() {
    return (<iframe className={'VR'} allow="vr" width= '100%' height= '100%'
```

```
            data-x={this.state.position[0]} data-y={this.state.position[1]} data-z={
                this.state.position[2]}
            src={'VR/index.html?id=' + this.props.id + "&visor=false"} id="receiver"
                />);
    }
    componentDidMount() {window.addEventListener("message", this.receiver);}
    componentWillUnmount() {window.removeEventListener('message', this.receiver);}
```

El estado sólo guarda la posición que le envía la aplicación de realidad virtual porque el identificador, estado del *plugin* y las marcas puestas ya se pasan desde el fichero del *plugin* al componente en forma de *props*. Para poder escuchar el evento de llegada de mensajes y gestionarlos, el componente debe añadir un escuchador de eventos (*eventListener*) antes de montarse, que apunte a la función que gestiona la conexión, y eliminarlo cuando se vaya a desmontar. Una vez establecida la conexión entre el *iframe* y Ediphy, la función que se encarga de enviar las actualizaciones hechas en la barra de herramientas es *toolbarUpdateValue(this.props)*. Para invocarla, el componente necesita implementar el método *componentWillReceiveProps(nextProps)* para que cada vez que se modifique el estado (se edite algo en la barra de herramientas) o se añada una marca, se llame a la función de que envía la actualización.

```
componentWillReceiveProps(nextProps) {
   if (JSON.stringify(this.props.state) !== JSON.stringify(nextProps.state)) {this.
       toolbarUpdateValue(nextProps);}
   if (JSON.stringify(this.props.marks) !== JSON.stringify(nextProps.marks)) {this.
       toolbarUpdateValue(nextProps);}
}
```

Para deshabilitar funcionalidades como reproducir o parar el audio ambiente, o lanzar acciones de las marcas, en la *query* que acompaña al fichero fuente con el que enlaza el *iframe* se le ha añadido el parámetro "visor"que indica a la aplicación con un valor booleano si estas funcionalidades deben o no habilitarse.

Configuración del *plugin* en modo visor(*VirtualReality.js*)

En el caso del visor, la única función que se implementa es para renderizar el componente de realidad virtual. Esta desagregación de componente y *plugin* en dos ficheros distintos se debe a la misma razón que en el editor.

```
    getRenderTemplate: function(state, props) {
        return <VirtualRealityPlugin state={state} id={props.id}
        marks={props.marks} onMarkClicked={props.onMarkClicked}/>;}
```

Sólo cambia respecto al componente para el editor en el paso del método *onMarkClicked()*, que tienen todos los *plugins* marcados como enriquecidos y que se utiliza para gestionar las acciones que lanzan las marcas al ser activadas.

Componente React para el visor (*VirtualRealityPlugin.js*)

Las diferencias entre ambos componentes son mínimas. Igual que en el editor, los métodos que se implementan en este fichero tienen como objetivo renderizar el *iframe* y gestionar la conexión. El atributo visor en este caso se indica que es verdadero, de manera la aplicación de React 360 habilita el control del sonido ambiente y el lanzamiento de acciones que se hayan especificado en las marcas creadas. Sin embargo, en este caso no se guardan las coordenadas de la posición a la que está mirando el usuario ya que no es necesario conocerlas para ninguna de las funcionalidades del *plugin* en el visor. Como se vio en el apartado de comunicación, en lugar de POSITION en el visor lo que se gestiona es MARK, para lanzar desde Ediphy, conociendo los datos de la marca que ha seleccionado el usuario, la acción correspondiente. La última diferencia con el componente del editor es que no se necesita el método *componentWillReceiveProps(nextProps)*, pues este sólo se ejecuta cuando hay cambios en la aplicación debido a la edición (modificando los valores de la barra de herramientas). Es decir, en el visor sólo es necesario conocer el estado de la aplicación con los cambios que se hayan editado, el identificador del *plugin* para poder comunicarse correctamente y la recepción de mensajes MARK para lanzar las acciones correspondientes.

4.3.4. Funcionalidad ajustada al contexto Ediphy - React 360

En esta sección se detalla el código [29] que ha sido implementado para desarrollar cada una de las funcionalidades de la aplicación. En todos ellos, se explica tanto la parte correspondiente a Ediphy (barra de herramientas) como la de la aplicación.

Imagen de fondo

Una de las funcionalidades básicas del *plugin* es cambiar la imagen de fondo de la escena. Para ello, se deben añadir elementos a la barra de herramientas que permita

al usuario escoger la imagen que quiere. Esta información se enviará a la aplicación de realidad virtual, que le indicará al componente de fondo (*Backgroud.js*) la imagen que debe renderizar.

VirtualReality.js* (Editor): *getToolbar() Para editar la imagen de fondo se ha añadido a la barra de herramientas un acordeón llamado *Background*, que contiene dos botones.

```
basic: {
       __name: "Background",
       icon: 'crop_original',
       buttons: {
           imagenBack: {
               __name: '',
               type: 'select',
               value: state.imagenBack,
               options: ['Elige un fondo...', '360_world.jpg', 'pano-planets.jpg', '
                   pano-nature.jpg', 'pano-nature2.jpg', 'pano-nature3.jpg', 'pano-
                   boom.jpg', 'pano-people.jpg'],
           },
           urlBack: {
               __name: 'Buscar entorno',
               type: 'external_provider',
               accept: "image/*",
               value: state.urlBack,
               autoManaged: false,
}}}
```

El primer botón, de tipo *select*, ofrece al usuario una lista de imágenes predefinidas con las que ya cuenta la aplicación. El segundo, es un *external_provider*, utiliza un componente de Ediphy que implementa las API de repositorios como Flickr o Vish para obtener imágenes (se indica que deben aceptarse sólo archivos de imagen), además de permitir la carga de archivos locales.

index.js El componente encargado de la selección de fondo, *Background.js*, depende del componente principal, *'Ediphy360'*, que se crea al generar el proyecto y se aloja en el fichero *index.js*. El único método que implementa es el *componentWillReceiveProps(nextProps)*.

```
componentWillReceiveProps(nextProps) {
  if ( nextProps.imgBack !== this.props.imgBack ) {
    Environment.setBackgroundImage(asset(nextProps.imgBack), {format: nextProps.
        format});
  } else if ( nextProps.urlBack !== this.props.urlBack ){
    Environment.setBackgroundImage(nextProps.urlBack, {format: nextProps.format});
  }
}render() {return null}
```

Background.js no renderiza nada directamente, sino que utiliza el módulo *Environement* de React 360 para poner un fondo nuevo cada vez que cambian los datos de entrada que recibe. Se han tenido que utilizar dos condiciones debido a que la sintaxis para un archivo local y para un archivo pasado mediante su URL, es diferente.

El audio ambiente

La otra funcionalidad básica de esta aplicación permite habilitar audio ambiente. Para ello, desde la barra de herramientas se puede elegir si mostrar los botones de control del audio. En caso de que se elija, en el visor los botones quedarán habilitados y el usuario que consuma el recurso podrá decidir si reproducirlo o pararlo. Dado que ya se ha explicado cómo funciona esta comprobación de estar o no en el visor, ahora se verá cómo está construida esta funcionalidad.

VirtualReality.js* (Editor): *getToolbar() En el acordeón de configuración se ha añadido un botón de tipo *checkbox* para indicar si deben mostrarse o no los botones de control del audio.

```
__name: "Configuration",
icon: 'build',
buttons: {
        audioBack: {
        __name: 'Audio ambiente',
        type: 'checkbox',
        checked: state.audioBack,}
```

client.js En *client.js* deben indicarse los componentes React que se renderizarán en las superficies o en las localizaciones que se hayan creado. En el caso del control de audio, los botones destinados a ello se renderizan en la superficie disponible por defecto, a la cuál se le asocia el componente principal *Ediphy 360* (en *index.js*).

```
const sup = r360.getDefaultSurface();
sup.setShape(Surface.SurfaceShape.Flat);
sup.resize(200, 90);
sup.setAngle(0, -0.4);
r360.renderToSurface(r360.createRoot('Ediphy360', { /* initial props */ }),sup);
```

index.js Para renderizar estos controles en la superficie, se utiliza un componente básico de React 360, el <*VrButton*>, el cuál se utiliza para envolver y proporcionar una función que se lanza al pulsar el contenido, un componente <*Text*> que índica la acción.

```
render() {
     return (
       <View style={styles.panelForControls}>
         <Background imgBack={this.state.imgBack} urlBack={this.state.urlBack}
             format={this.state.format} />
         {this.state.showAudio ? (
           <View style={styles.controls}>
           <VrButton onClick={this._playAudio} style={styles.button}>
               <Text style={styles.buttonText}>{'Play'}</Text>
           </VrButton>
           <VrButton onClick={this._stopAudio} style={styles.button}>
               <Text style={styles.buttonText}>{'Stop'}</Text>
           </VrButton>
         </View>) : null}
       </View>);}
```

La función a la que se llama cuando el usuario selecciona el botón de reproducir utiliza el módulo nativo *AudioModule* en caso de estar siendo ejecutada desde el visor. Todas las reproducciones de audio (audio ambiental, efectos de sonido únicos y audio especializado) se controlan en React 360 a través de este módulo nativo. En el caso de esta aplicación, se utiliza el audio ambiental y para activarlo es necesario el método *playEnvironmental()* que expone el módulo. Por defecto está en bucle y en la configuración se deben especificar los valores de la ruta al recurso de audio y el volumen. También pueden cambiarse los valores de reproducción una vez esta ha empezado (deshabilitar en bucle por defecto, silenciar el audio, etc). Para parar el audio, simplemente se llama al método *stopEnvironemental()* del módulo.

El carrusel de imágenes

Otra funcionalidad que tiene esta aplicación es la de presentar un panel central con las imágenes que decida cargar el usuario a modo carrusel. Para ello, desde la barra de herramientas de Ediphy se puede seleccionar si habilitar o no esta función y cargar las imágenes que se quieran proyectar (máximo 10). En el visor, el usuario puede usar los controles laterales para avanzar por el carrusel.

VirtualReality.js* (Editor): *getToolbar() En el acordeón de configuración se ha añadido un botón de tipo *checkbox* para indicar si debe mostrarse o no el carrusel. En caso de ser activado, se despliegan el resto de los botones: uno para elegir el número de imágenes que se quieren cargar y otro de tipo *external_provider* por cada imagen cargada.

```
let urlPanels = {};
   for (let i = 0; i < state.numberOfPictures; i++) {
       urlPanels["urlPanel" + i] = {
           __name: 'URL ' + (i + 1),
           type: 'external_provider',
           accept: "image/*",
           value: state["urlPanel" + i],
           hide: !state.showPanel,
           autoManaged: false,
   };}/*configuración de la barra de herramientas del resto de     funcionalidades*/
       showPanel: {
           __name: 'Panel Aux',
           type: 'checkbox',
           checked: state.showPanel,
       },numberOfPictures: {
           __name: 'Number of pictures',
           type: 'number',
           min: 1,
           max: 10,
           hide: !state.showPanel,
           value: state.numberOfPictures,
       },...urlPanels,
```

client.js Para este componente, llamado Proyector, se crea otra superficie durante la inicialización, en el tiempo de ejecución.

```
import {Location, ReactInstance, Surface} from 'react-360-web';
const photosPanel = new Surface(700, 600, Surface.SurfaceShape.Cylinder);
//Código para el resto de funcionalidades
r360.renderToSurface(r360.createRoot('Proyector'),photosPanel,);
```

Al ser un componente raíz, es necesario registrarlo con *AppRegistry.registerComponent()* en el fichero *index.js* para poder asociarlo con la superficie.

ProyectorComponente.js Este componente es el encargado de renderizar en la superficie a la que está asociado las imágenes seleccionadas en la barra de herramientas por el usuario. Tal y como se explicó en el apartado del *ConexionModule*, el estado se actualiza con los datos que le envía Ediphy. En el caso de que el control esté activado, *showPanel* será verdadero y se renderizará el proyector en la superficie. En *arrayImgs* se guardan las direcciones de las imágenes proporcionadas en la barra de herramientas, y en *currentImg*, la imagen que se esté mostrando en el panel (por defecto, la primera del conjunto).

```
this.state = {arrayImgs: [],keySelected: undefined,currentImg: undefined,
    showPanel: false,};
```

Este componente dispone de dos imágenes cuyas funciones son cambiar en el estado la imagen que está seleccionada para que se renderice la anterior o la siguiente del carrusel (avanzar, retroceder en el carrusel).

```
onPrevClick(){
   let key = this.state.keySelected;
   if(key != undefined){if(key == 0){key = this.state.arrayImgs.length-1;}else{ key
       = key-1;}
   }else{return null;}
   this.setState({keySelected: key, currentImg: this.state.arrayImgs[key].currentImg
       });}
 onNextClick(){
   let key = this.state.keySelected;
   if(key != undefined){if(key == this.state.arrayImgs.length-1){key = 0;}else{key =
        key+1;}
   }else{return null;}
   this.setState({keySelected: key, currentImg: this.state.arrayImgs[key].currentImg
       });}
 render() {
   if (!this.state.showPanel) {return null;}else{
         return (
           <View style={styles.flatpanel}>
             <View style={styles.controls}>
               <VrButton onClick={this.onPrevClick} disabled={this.state.arrayImgs.
                   length < 1}>
                 <Image style={styles.iconPrev} source={asset('icons/prev.png')} />
               </VrButton>
               {this.state.currentImg ? (
                <Image style={styles.img} source={{uri: this.state.currentImg}} />
               ) : <View style={styles.img}></View>}
               <VrButton onClick={this.onNextClick} disabled={this.state.arrayImgs.
                   length < 1}>
                 <Image style={styles.iconNext} source={asset('icons/next.png')} />
               </VrButton>
             </View>
           </View>
         );}}
```

Marcas en el espacio

La funcionalidad más compleja del *plugin* es la de permitir al usuario posicionar marcas tridimensionales por el espacio, desde las cuales se pueda enlazar con otro contenido del curso o mostrar un texto emergente. Para ello, se ha indicado que se trata de un *plugin enriquecido*, por lo que se han añadido a la configuración los métodos necesarios para manejar esta funcionalidad.

VirtualReality.js* (Editor): *parseRichMarkInput() Convierte las coordenadas del punto que selecciona el usuario en la zona habilitada para *plugins enriquecidos*

(*dropableRichZone*) en el valor correspondiente en la aplicación, es decir, en coordenadas tridimensionales. El argumento es un array compuesto por 6 valores:

- *value[0]* es la coordenada x en píxeles relativa a la esquina superior izquierda de la *dropableRichZone*.
- *value[1]* es la coordenada y en píxeles relativa a la esquina superior izquierda de la *dropableRichZone*
- *value[2]* es el ancho de la zona en píxeles (coincide con el ancho del *iframe*).
- *value[3]* es el alto de la zona en píxeles (coincide con el alto del *iframe*).
- *value[4]* está en desuso.
- *value[5]* es el estado del *plugin*.
- *value[6]* es el identificador de la caja que contiene al *plugin*.

Para calcular la posición en la escena de realidad virtual, debe conocerse la rotación actual con la que está observando el usuario, y esta la proporciona la aplicación a Ediphy a través de postmessage, enviando POSITION como parámetro de acción. Ppara transformar de píxeles a metros se debe conocer cuantos metros se están renderizando en la pantalla, y como la relación de aspecto es fija, este valor es constante. Así, se ha determinado que se observan 10 metros de ancho y 5 de alto. Con todo lo anterior y haciendo las transformaciones trigonométricas necesarias, se obtiene el valor de la posición en la que debe colocarse la marca en la escena tridimensional.

```
parseRichMarkInput: function(...value) {
          let xPix = (value[0] - value[2] / 2);
          let yPix = -(value[1] - value[3] / 2);
          const R = 4;
          let xMet = -xPix / value[2] * 1.75 * Math.PI / 3;
          let yMet = yPix / value[3] * 2 * Math.PI / 3 * value[3] / value[2];
          let vrApp = document.querySelector('#box-' + value[6] + ' .VR');
          let ang = [vrApp.getAttribute('data-x'), vrApp.getAttribute('data-y'),
              vrApp.getAttribute('data-z')];
          ang = ang.map(a => a * Math.PI / 180);
          let x360 = -R * Math.sin(ang[1] + xMet) * Math.cos(ang[0] + yMet);
          let y360 = R * Math.sin(ang[0] + yMet);
          let z360 = -R * Math.cos(ang[0] + yMet) * Math.cos(ang[1] + xMet);
          let finalValue = x360.toFixed(2) + "," + y360.toFixed(2) + "," + z360.
              toFixed(2);
          return finalValue;
      }
```

VirtualReality.js* (Editor): *getDefaultMarkValue() Calcula el valor por defecto de la marca. Esto es necesario para cuando se crea desde la barra de herramientas en lugar de posicionando sobre la ventana del *iframe* y si no es un valor constante, como es el caso, ya que dependerá de la posición a la que esté mirando el usuario.

```
getDefaultMarkValue(state, id) {
            let x, y, z = 0;
            const R = 4;
            let vrApp = document.querySelector('#box-' + id + ' .VR');
            let ang = [0, 0, 0];
            if (vrApp) {
                ang = [vrApp.getAttribute('data-x'), vrApp.getAttribute('data-y'),
                    vrApp.getAttribute('data-z')];}
            ang = ang.map(a=>a * Math.PI / 180);
            x = -R * Math.sin(ang[1]) * Math.cos(ang[0]);// + x;
            y = R * Math.sin(ang[0]); // + y;
            z = -R * Math.cos(ang[0]) * Math.cos(ang[1]);
            let finalValue = x.toFixed(2) + "," + y.toFixed(2) + "," + z.toFixed(2);
            return finalValue;}
```

VirtualReality.js* (Editor): *validateValueInput() Se encarga de validar la entrada de un usuario cuando introduce el valor de la marca manualmente desde la barra de herramientas. Por ejemplo, si el usuario sólo introduce una coordenada, más de 3 o alguna con formato no válido para este *plugin* (% por ejemplo).

```
validateValueInput: function(value) {
    let regex = /(^-*\d+(?:\.\d*)?),(-*\d+(?:\.\d*)?),(-*\d+(?:\.\d*)?$)/g;
    let match = regex.exec(value);
    if (match && match.length === 4) {
        let x = Math.round(parseFloat(match[1]) * 100000) / 100000;
        let y = Math.round(parseFloat(match[2]) * 100000) / 100000;
        let z = Math.round(parseFloat(match[3]) * 100000) / 100000;
        if (isNaN(x) || isNaN(y)) {return { isWrong: true, message:i18n.t("
            VirtualTour.message_mark_xy") };}
        value = x + ',' + y + ',' + z;}else {return { isWrong: true, message: i18n.t(
            "VirtualTour.message_mark_xy") };}return { isWrong: false, value: value
            };}
```

client.js La representación visual de las marcas es en la aplicación son objetos 3D. Estos no pueden agregarse a las superficies ya que están restringidas a contenido 2D. Para montar un árbol de componentes 3D y renderizarlo en la escena, debe hacerse sobre una Localización (*Location*). Estas representan orígenes de coordenadas en el espacio físico y de forma relativa a ellas se puede situar cualquier objeto 3D en el mundo virtual.

```
r360.renderToLocation(r360.createRoot('Marks'),r360.getDefaultLocation());
```

Al ser un componente raíz, es necesario registrarlo con *AppRegistry.registerComponent()* en el fichero *index.js* para poder asociarlo con la Localización.

Marks.js Este componente es el encargado de renderizar, respecto a la Localización a la que está asociado, las marcas que añada el usuario. Para ello, guarda en el estado el array de marcas que le envía Ediphy a la aplicación. En su método *render*, recorre el array de marcas utilizando la función *map()* y para cada elemento genera un nuevo componente *AnimatedMark* al que pasa los datos de la marca y la función con la que llama a al módulo nativo de la conexión con Ediphy, para el caso en el que necesite que se lance alguna acción asociada con la marca desde el propio Ediphy.

```
render() {
   let marks = [];
   for (let mark in this.state.marks) {marks.push(this.state.marks[mark])}
   if(marks === []){return null}else{let marcas =  marks.map((mark,key)=>{
     return <AnimatedMark key={key} {...mark} onClick={this.sendMarkEvent}/>})
     return <View>{marcas}</View>
   }}
```

AnimatedMark.js Se encarga de renderizar el objeto de la marca con la información que esta trae consigo desde Ediphy, como son las coordenadas, el color y, en el caso de ser de tipo emergente, el texto asociado. Para ello, lo primero que hace es obtener las coordenadas como números de la cadena de texto en la que vienen. Dependiendo del valor de estas, si son emergentes el texto deberá rotar una cierta cantidad de grados para que el usuario pueda verlo (que no quede perpendicular a él), así que se calcula el valor aproximado de esta rotación. A continuación, se renderiza dentro de un <*VrButton*> el objeto 3D que se ha editado y alojado en el fichero *modelHueco.obj*.

```
render() {
    let coorX = Number(this.props.value.split(",")[0]);
    let coorY = Number(this.props.value.split(",")[1]);
    let coorZ = Number(this.props.value.split(",")[2]);
    let zTextPos = 1;
    let yRot =  -90+180/Math.PI*Math.tan2(-coorX,-coorZ);
      return (
        <VrButton style={{flex: 1,flexDirection: 'row',justifyContent: 'space-between
            ',backgroundColor:'transparent',position: 'absolute',transform: [ {
            translate: [coorX,coorY,coorZ]}],
      }} onClick={this.handleMarkClick}>
           <Text style={{fontSize: 0.2,width: 2,color: 'black',backgroundColor: '
               white',transform: [{translate:[0 ,0.8, zTextPos]},{rotateY: yRot}],
               opacity: (this.state.show ? 1:0)}}>
            {this.props.connection}
```

```
            </Text>
           <AnimatedEntity style={{color: this.props.color,transform: [{scale: 0.01},{
               rotateY: this.rotation}]}}
             source={{obj: asset('icons/modelHueco.obj'),}}/>
         </VrButton>)
```

Para darle mayor dinamismo a la escena, se ha utilizado el módulo *Animated* para implementar una función que haga a la marca rotar sobre sí misma desde el momento en el que se coloca en la escena.

```
spin = () => {
  this.rotation.setValue(0);
  Animated.timing(this.rotation, {toValue: 360, easing: Easing.linear, duration:
      2500}).start((animation) => {
    if (animation.finished) {this.spin();}});};
```

La función encargada de comunicarse con Ediphy para las marcas que enlazan con otro contenido llama a su equivalente en el componente que agrupa a todas las marcas y este utiliza el módulo nativo de conexión que ya se ha explicado. Si es una marca de tipo *pop-up*, se utiliza la función *enVisor()* del mismo módulo nativo para ver si debe o no mostrarse el texto que lleva asociado cuando el usuario selecciona la marca.

Capítulo 5

Resultados y validación

5.1. Resultados

Este trabajo ha conseguido integrar una aplicación de realidad virtual que puede ser editada con fines educativos en la herramienta Ediphy. Una vez completado el desarrollo, el nuevo *plugin* permite a los usuarios crear paseos de aprendizaje secuenciados en tres dimensiones, con la opción de cambiar el escenario y añadir sonido de ambiente. La nueva funcionalidad permite también añadir a los escenarios un carrusel de imágenes y colocar puntos calientes (marcas) sobre la escena tridimensional que permitan navegación dentro o fuera del recurso y el lanzamiento de mensajes emergentes.

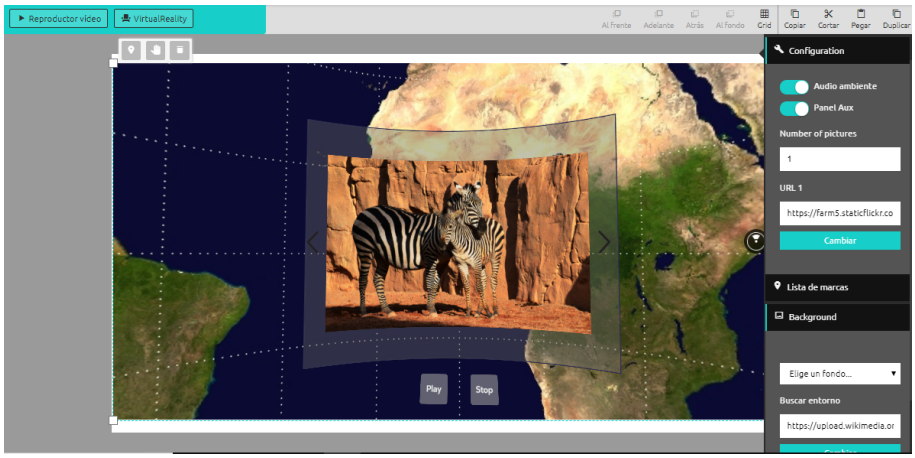

Figura 9 : Habilitar audio y carrusel

En la figura anterior puede verse la barra de herramientas de Ediphy para este *plugin*. Cuenta con los botones que habilitan el audio ambiente y el carrusel, los campos para buscar o introducir las direcciones de las imágenes del carrusel y las opciones para seleccionar el fondo. Las marcas pueden situarse en la escena utilizando el botón .ªñadir marca"de la barra de herramientas (las coloca por defecto en la posición central del punto

al que se esté mirando) o con un botón a modo *shortcut* que aparece en la caja del *plugin*, y que permite señalar sobre el propio escenario dónde se quiere colocar la marca.

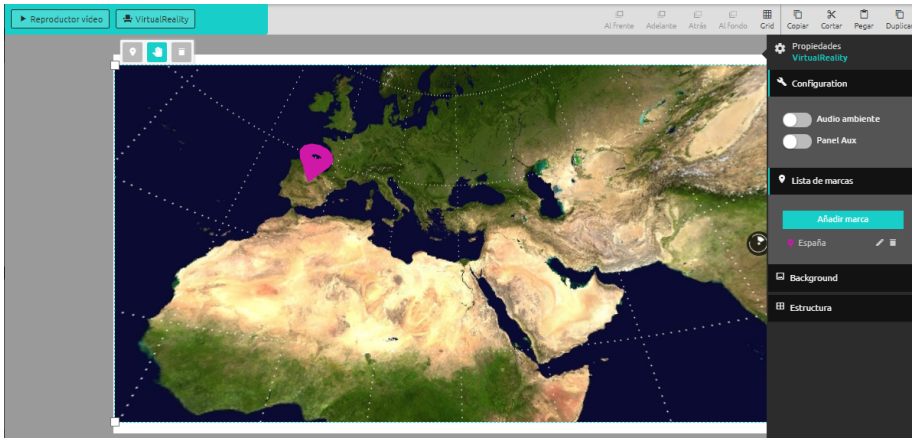

Figura 10 : Posicionar marcas por el entorno

Al pasar al modo de previsualización, se pueden activar las marcas para leer su contenido en el caso de que sean de tipo *pop-up* o bien para lanzar la diapositiva, documento o contenido externo con el que se ha enlazado.

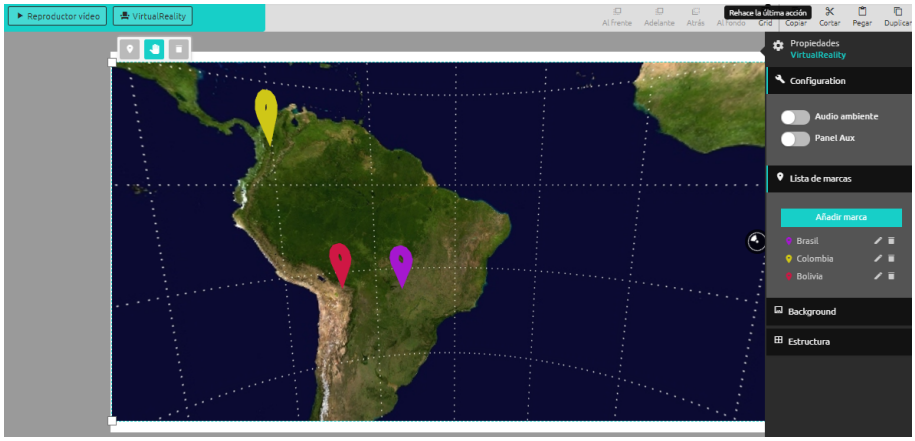

Figura 11 : Información en las marcas

5.2. Pruebas realizadas

5.2.1. Test de requisitos del *plugin*

Para comprobar la correcta integración de la nueva funcionalidad en Ediphy, se siguieron desde el inicio del proyecto los pasos definidos por el equipo de desarrollo para adaptarse al cumplimiento de los requisitos mínimos que debe tener una nueva aportación de código. Para ello, se partió de la creación del *plugin* mediante la API definida en la documentación, y esto permitía asegurar una estructura común al resto de *plugins* de la aplicación. Durante el desarrollo se tuvieron en cuenta las funciones a las que se debía llamar, tratando en todo momento de respetar los principios de componentización que marca Ediphy en la construcción de sus módulos. Tenidos en cuenta los aspectos manuales del proceso de implementación, se cerró la validación de la integración recurriendo a la batería de test con la que ya cuenta el entorno de desarrollo para validar los requisitos exigidos en la agregación de nuevas funcionalidades. Los test en Ediphy están elaborados con Jest [30] una librería utilizada por Facebook para testear código JavaScript, incluidas las aplicaciones de React. Permite construir test unitarios trabajando con *matchers* personalizados, crear *mocks* o comprobar instantáneas de componentes visuales de manera sencilla. Para pasar la validación de los test definidos en Ediphy, se debe utilizar el comando *zarn test"* o *zarn test:verbose"*, según el nivel de detalle que se quiera obtener en la respuesta.

5.2.2. Escalabilidad

Las librerías de realidad virtual requieren más recursos de hardware que otras menos exigentes en relación a los gráficos. Por este motivo, se hacen necesarias unas pruebas de carga mínimas que permitan comprobar en el navegador la escalabilidad de la aplicación React 360 propuesta. Una forma sencilla de hacerlo es comprobar el aumento de gasto de recursos al añadir instancias del *plugin* en el navegador. Se ha comprobado que al superar cierto número de instancias, dependiendo de las especificaciones del hardware en el que se ejecuten, la aplicación deja de funcionar correctamente por falta de recursos.

Capítulo 6

Conclusiones y líneas futuras

6.1. Conclusiones

Con la nueva funcionalidad propuesta, los usuarios de Ediphy disponen de más opciones a la hora de diseñar sus recursos educativos, ya sea para enseñar conceptos o presentar información de un modo más atractivo. De esta forma, se amplían las posibilidades de la herramienta, que se une además a la corriente tecnológica que poco a poco está reinventando los métodos educativos tradicionales.

Aprovechando la potencia de la realidad virtual para que los recursos educativos integren entornos inmersivos y dinámicos, y basado en tecnologías de vanguardia, este proyecto deja sentadas las bases para futuras ampliaciones y nuevas maneras de aprovechar los recursos que ofrece la realidad virtual en una herramienta como Ediphy.

El desarrollo de la librería de realidad virtual que hemos utilizado está en continua evolución, añadiendo funcionalidades y mejorando las que ya tiene. Se lanzó a la comunidad hace apenas dos meses y ya existen cientos de desarrolladores que están utilizándola para adaptar sus aplicaciones de realidad virtual antiguas o para seguir creando otras nuevas, siempre con el objetivo de ofrecer el contenido web de una manera distinta, que atrae y atrapa a los usuarios. Sin duda, el uso de este tipo de librerías marcará un antes y un después en la concepción sobre la navegación por internet que tenemos hoy en día.

Otra de las vertientes relacionadas con este trabajo que se pueden abordar en el futuro es la del estudio de la escalabilidad. Ya se ha probado que el aumento de instancias afecta al correcto funcionamiento de la aplicación, pero podría hacerse un estudio exhaustivo que determinase si es recomendable limitar la cantidad de estas que pueden tener los usuarios activadas a la vez.

6.2. Líneas futuras

Con el trabajo desarrollado, se ha diseñado e implementado la comunicación entre Ediphy y una aplicación en React VR. Esta labor establece una base desde la que se puede comenzar para el desarrollo de las ideas que en el futuro se quieran implementar en la línea de los entornos de realidad virtual. Existen numerosas aplicaciones que pueden desarrollarse utilizando la realidad virtual.

Un ejemplo podría ser la ampliación de la definición de los paseos propuestos en este trabajo, de forma que el usuario pueda moverse entre distintos entornos 360, creando así ambientes más dinámicos, pudiendo generar entornos muy cercanos al aprendizaje basado en juegos[31].

Otra posible línea de mejora puede ser la introducción de objetos tridimensionales complejos dentro del espacio de realidad virtual, poder manejarlos y verlos desde diferentes perspectivas y que aporten una mayor información al usuario. Relacionado con las funcionalidades con las que cuenta ahora el *plugin*, otra de las líneas en las que se puede seguir trabajando es en la precisión de la colocación de las marcas en la escena. Esto implicaría una profundización en todos los elementos relacionados con la transformación de coordenadas, donde habría que atender a la gestión de numerosos parámetros, como el ángulo de visión, la resolución en píxeles de la pantalla y la profundidad de las marcas.

Otra posible línea podría centrarse en la mejora del *plugin* enriqueciendo el carrusel de imágenes de forma que sea capaz de admitir diferentes tipos de recursos, como vídeos o gráficos; y poder cambiar su posición, para decidir en qué lugar de la escena debe mostrarse.

Anexos

Anexo I

Análisis de Impacto

Como se establece en los requisitos de las acreditaciones internacionales EUR-ACE y ABET, se incluye en este trabajo un anexo en el que se reflexiona sobre el posible impacto y las responsabilidades relacionadas con el mismo. La nueva aplicación de realidad virtual ayudará a quienes accedan a la herramienta web para crear un curso, permitiéndoles utilizar de manera sencilla e intuitiva los beneficios que ofrece esta tecnología.

A. Impacto social

Este trabajo utiliza dos conceptos de la tecnología que están actualmente aumentando su presencia en la sociedad: el *e-Learning* y la realidad virtual. Dentro del primero, se pueden enmarcar distintos tipos de contenido, pero lo cierto es que los resultados que ofrecen los recursos interactivos a la hora de mejorar el proceso de aprendizaje son, en muchas ocasiones, mejores que con recursos tradicionales. Ediphy cuenta con una serie de *plugins* que ofrecen a los usuarios distintas posibilidades para enriquecer sus cursos educativos, pero ninguno conseguía ofrecer un grado de inmersión e interactividad como el que permite el uso de la realidad virtual, ya que transportar a los usuarios a entornos que simulen una realidad concreta diseñada por el editor, es precisamente la razón de ser de esta. Por lo tanto, con la nueva herramienta de realidad virtual de Ediphy, la sociedad consumidora de este tipo de cursos podrá disfrutar ahora de una nueva forma de aprender conceptos, gráficamente más atractiva y que ayuda a la comprensión de los mismos.

B. Impacto medioambiental

Uno de los mayores problemas medioambientales con los que la sociedad debe lidiar actualmente es la generación masiva de residuos. Muchos de estos residuos se producen en acciones cotidianas, entre las que podría encontrarse el estudio de un tema concreto a través de libros, apuntes o, en general, recursos físicos. Gracias a herramientas como Ediphy, este tipo de recursos pueden sustituirse por sus equivalentes *on-line*, reduciendo considerablemente el gasto de recursos físicos como el papel. Además, el *plugin* de realidad virtual concretamente también ayuda a fomentar la educación a distancia, ya que es posible arreglar el inconveniente del espacio físico. Con esto, se reduce la contaminación producida por los desplazamientos a los lugares tradicionales de enseñanza.

C. Impacto económico

Relacionado con el impacto medioambiental se encuentra el ahorro económico que supone tanto para las entidades consumidoras de la herramienta Ediphy como para sus propios desarrolladores el hecho de necesitar una cantidad de recursos físicos mucho menor que con herramientas tradicionales. Al ser un proyecto de código libre, el beneficio económico que genera el desarrollo de Ediphy viene marcado por su capacidad de adaptación a clientes específicos. Para ello, la herramienta debe contar con un mantenimiento continuo que se adapte a los cambios y avances tecnológicos, así como a los requisitos que surjan por parte de los usuarios. Este mantenimiento debe contemplarse a la hora de estudiar la viabilidad económica del proyecto, ya que supone costes de mantenimiento y actualización de la herramienta.

D. Responsabilidad ética y profesional

En el caso del proyecto de realidad virtual no se ha encontrado aplicación para este tipo de impacto.

Anexo II

Presupuesto económico

El presupuesto de este proyecto se desglosa, según la naturaleza de los costes implicados, en 5 partes: costes materiales, costes de mano de obra, costes por licencia, costes por impuestos y otros costes. En un último apartado se muestra el coste total.

A. Costes materiales

El material necesario para la realización de este trabajo, tanto para la investigación como para el desarrollo de la aplicación de realidad virtual, ha sido un ordenador con sistema operativo Windows 10, procesador Intel Core i7-4510U, 8GB de memoria RAM y 1 TB de almacenamiento, que actualmente tiene un precio de mercado de aproximadamente 600 €. La vida útil de un ordenador de estas características es de 4 años, por lo que, teniendo en cuenta que el desarrollo del proyecto ha durado casi 6 meses, se ha hecho un uso del 12.5 % de su vida útil. Traducido a costes de material supone 75 €.

B. Costes de mano de obra

Los costes de mano de obra se han calculado teniendo en cuenta las horas empleadas en las distintas fases del desarrollo del proyecto y el salario medio. Según el COIT, el sueldo medio de un Ingeniero de Telecomunicación es de 52.711 €brutos al año. Sin embargo, este dato no refleja la condición exacta de la titulación que aplica a este proyecto, y como el COITT no ofrece un dato equivalente en su página web para el caso de los Ingenieros Técnicos, el salario medio ha sido calculado teniendo en cuenta lo que ofrecen las empresas del sector a un recién titulado en un grado de ingeniería técnica como es el Grado en Ingeniería de Tecnologías y Servicios de la Telecomunicación.

Fase	Dedicación (horas)
Investigación	90
Desarrollo	160
Validación	5
Redacción de la memoria	60
Total	315

Tabla 1 : Tiempo dedicado a realizar el proyecto

El salario medio para un trabajador con las características anteriores es de 24.000 €/año. Esto son 1920 horas de trabajo si se tiene una jornada laboral de 8 horas/día y 20 días de vacaciones. Por lo tanto, el resultado es un sueldo de 12.5€/hora.

Horas trabajadas	Coste de mano de obra total
300	3750 €

Tabla 2 : Coste de mano de obra

C. Costes por licencia

El software empleado es gratuito y las licencias son de código libre, por lo que no existen costes adicionales por licencias.

D. Costes por impuestos

Vender el software que se ha implementado implicaría tener que añadir los impuesto derivados de la venta, que actualmente son de un 21 % sobre el precio final.

E. Otros costes

Debido a las características del proyecto, es necesario tener en cuenta también el coste relacionado con el consumo eléctrico de la máquina utilizada para su desarrollo. Se estima que un ordenador de las características que se han comentado en el apartado de

costes materiales consume 1 kWh. De media, el coste del kWh es de 0.12 €/kWh. Por lo que quedaría un coste de 132 €.

F. Coste total

Concepto	Coste (€)
Costes materiales	75
Costes de mano de obra	3750
Costes de licencias	0.00
Otros costes	132
Impuestos	830.97
Total	4787.97

Tabla 3 : Total de costes

Bibliografía

[1] J. Valverde, E. López, M. Garrido, and D. Díaz, "Análisis descriptivo y de carácter pedagógico sobre programas informáticos para la implementación de plataformas de aprendizaje electrónico," E-LEARNING, 2001.

[2] "Javascript." [Online]. Available: https://www.javascript.com

[3] "Javascript." [Online]. Available: https://developer.mozilla.org/es/docs/Learn/JavaScript

[4] "Ecmascript." [Online]. Available: https://tc39.github.io/ecma262/

[5] "Api twitter." [Online]. Available: https://developer.twitter.com/en/docs.html

[6] "Api google maps." [Online]. Available: https://developers.google.com/maps/documentation/javascript/reference/3/

[7] "React." [Online]. Available: https://reactjs.org/

[8] "Jsx." [Online]. Available: https://reactjs.org/docs/introducing-jsx.html

[9] "Redux." [Online]. Available: https://es.redux.js.org/

[10] "Flux." [Online]. Available: http://facebook.github.io/flux/

[11] "Webpack." [Online]. Available: https://webpack.js.org/

[12] "React native." [Online]. Available: https://facebook.github.io/react-native/

[13] J. G. Maldonado, "Aplicaciones de la realidad virtual en psicología clínica," Aula médica psiquiátrica, vol. 4, no. 2, pp. 92–126, 2002.

[14] "Google cardboard." [Online]. Available: https://vr.google.com/cardboard/

[15] "Oculus rift." [Online]. Available: https://www.oculus.com/rift/

[16] "Samsung gear vr." [Online]. Available: https://www.oculus.com/gear-vr/

[17] "Webgl." [Online]. Available: https://developer.mozilla.org/es/docs/Web/API/WebGL_API

[18] "Webvr." [Online]. Available: https://webvr.info/developers/

[19] "Three.js." [Online]. Available: https://threejs.org/

[20] "React vr." [Online]. Available: https://facebook.github.io/react-vr/

[21] "Oculus vr." [Online]. Available: https://www.oculus.com/

[22] "Dispositivos oculus." [Online]. Available: https://support.oculus.com/guides/

[23] "React 360." [Online]. Available: https://facebook.github.io/react-360/

[24] "El objeto window." [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Window

[25] "La función postmessage()." [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Window/postMessage

[26] "Ediphy." [Online]. Available: http://ging.github.io/ediphy/#/

[27] "Ediphy." [Online]. Available: https://github.com/ging/ediphy

[28] "Getting started with react vr." [Online]. Available: https://www.pluralsight.com/guides/getting-started-with-react-vr

[29] "Código de la aplicación de react 360." [Online]. Available: https://github.com/pagonzal13/React360

[30] "Jest." [Online]. Available: https://facebook.github.io/jest/

[31] E. R. Arteaga and V. T. Cosío, "Videojuegos y habilidades del pensamiento/videogames and thinking skills," RIDE Revista Iberoamericana para la Investigación y el Desarrollo Educativo, vol. 8, no. 16, pp. 267–288, 2018.