POLYTECHNIC UNIVERSITY OF MADRID

Higher Technical School of Telecommunication Engineering

CONTRIBUTION TO A HERRAMIENTAWEB OF Authorship RESOURCE E-LEARNING By integrating a component of VIRTUAL REALITY

DEGREE IN ENGINEERING AND TECHNOLOGY TELECOMMUNICATIONS SERVICES

Final Project

PAULA GONZÁLEZ GÓMEZ

Madrid, Spain
July, 2018

Superior Telematic Systems Engineering Department Technical

School of Telecommunication Engineering

CONTRIBUTION TO AUTHORSHIP HERRAMIENTAWEB RESOURCE E-LEARNING by integrating A virtual reality component

DEGREE IN ENGINEERING AND TECHNOLOGY TELECOMMUNICATIONS SERVICES

Final Project

Author: PAULA GONZÁLEZ GÓMEZ

Tutor: Lourdes Pascual Marcos

The rapporteur Enrique Arias Barra

July, 2018

President: ____

Vocal: ____

Secretary: ____

Alternate: ____

Performed the act of reading the defense and TFM day ______ from ____________ from ______ in Madrid, having obtained the quali fi cation ____________________________

President, The Secretary, Vocal,

Summary

Ediphy is an open source web tool for creating educational resources structure based on modules called plugins. Through an editing environment and aggregation of these plugins, users can create and preview resources created.

The objective of this Final Project is to expand the functionality of Ediphy by integrating a plug-in virtual reality. The new functionality gives users the ability to create environments 360Z with the image selected, enable ambient sound, and add a slideshow on the scene. In addition, the plugin allows you to add hot spots (marks) in three-dimensional space with which to include pop-up messages with explanatory content or navigation between the views of the resort.

To develop a first phase work research libraries raised on virtual reality and their possible applications in the context of Ediphy. The next phase of development focused on effective communication between the application of virtual reality and Ediphy that would allow the configuration of the virtual environment. Once communication is established, they were made both to implement native application modules virtual reality, as the implementation of the plugin and conguration in Ediphy. Finally, the validation phase allowed verifying the correct integration of new functionality added.

Keywords: Ediphy, plugins, multimedia, virtual reality, 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, users can aggregate Which in an editing environment to generate and preview the resources They are creating.

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

The fi rst 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 link Communication between the virtual reality application and Ediphy That allowed for the con fi guration of the virtual environment. Once esta connection was established, we Implemented the native virtual reality modules Within the application, as well as the plugin and Its con fi guration 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

Thanks

Ami tutor, Lourdes, for having said convinced that itself could get and help.

A Sonsoles, being always available and able to solve any doubt.

My parents, for having the confidence in me above everything and answer the phone always with a smile and the whole mood of the world.

My sister, being the person who best knows me and imposition of positive thinking.

To my eternal compilation of uni, Julian, because we are a team.

And my grandmother, having always lit a candle.

Thank you.

Index

Illustrations

Boards

We acronyms

API Applications Programming Interface

SUN Document Object Model

ECMA European Computer Manufacturers Association

SS6 ECMAScript

GPU Graphics Processing Unit, Processing Unit Graph co

JS JavaScript

MVC Model View Controller

PC Personal Computer, Personal Computer

RV Virtual reality

TIC Technology of the information and communication

UPM Polytechnic University of Madrid

URI Uniform Resource Identi fi er, identifier Uniform Resource

URL Uniform Resource Locator, Uniform Resource Locator

WebGL Web Graphics Library

WebVR Web Virtual Reality

3D Three dimensions

2D Two dimensions

Chapter 1

Introduction and objectives

Education in the last twenty years has undergone led transformation by information and communications technology (ICT) has changed the way in which educational content, affecting several areas, from the inclusion of contents and subjects are taught ICT-related, to the manner in which these contents are transmitted.

One of the most important terms in this transformation is the online training or e-learning, which is generally known by its English expression *e-Learning.* This concept refers to the processes of teaching and learning are carried out by using the Internet, and are characterized by physical separation between teachers and students. Because of its flexibility, this type of learning gives students the ability to self-manage the pace of learning, and to optimize the time spent on training. According to the analysis that make Valverde Lopez Garrido, Diaz and Rosenberg [1], virtual education offers numerous advantages such as greater richness of the learning process, increased motivation for learning, communication among educators, tracking the learning process, ease of updating content, cost reduction and easy access.

In this scenario, it is increasingly appropriate incorporating new tools and technologies that enable the dynamic creation of educational resources, to propose alternative ways of learning tailored to the characteristics of new media. Virtual reality, immersion qualities, is shown as a possible solution to extend the functionality of the *e-Learning.* As already mentioned, one of the biggest advantages of this type of education is to disaggregate space and time temporary teaching by the teacher to the student. Therefore, the possibility of building three-dimensional spaces that can become consumed in specialized viewers at any time or place, opens a new range of possibilities for the development of education through ICT.

The main objective of this work focuses on expanding the functionality of Ediphy, a web authoring tool and open source whose general purpose is the creation and editing of multimedia educational resources to help generate quality educational content for use in *e-Learning.* This improvement is focused on allowing users to use virtual reality as a support for the generation of educational content.

The modular structure that facilitates Ediphy is based contribution to the development with new features that can be added simply by creating minimum content units to which calls *plugins.*

Taking advantage of the facilities offered by the tool, it has incorporated a new *plugin*
that lets you add virtual reality scenarios the authors of resources created.

To achieve the objective set, this work has been developed in several phases, which have been fulfilling various objectives to reach the fi nal result. During the first phase of research libraries have studied virtual reality applications available and possible that these could have in a context like Ediphy. Then we designed a model of communication between application components and virtual reality *plugin* of Ediphy that would allow con fi gure from the toolbar the virtual environment. With this objective achieved, it was possible to start developing both the implementation of virtual reality modules as aspects related to the configuration of the *plugin* in Ediphy. Finally, during the validation phase it has verified the correct integration of new functionality added.

The new *plugin* of Ediphy allows users to choose between settings available in both specialized repositories as a local selection. In these environments, users can create learning walks using information marks positioned along all the selected virtual world. The results offered by this kind of immersion in retaining knowledge are undoubtedly better than with traditional methods, they help concentration and association of concepts by the student.

Episode 2

State of the art

2.1. Technologies

2.1.1. JavaScript

JavaScript (JS) [2] [3] is a lightweight language, interpreted and based on prototypes. It was designed by Brendan Eich in 1995 and currently is a registered trademark of Oracle Corporation. It is used in development of dynamic web pages to respond faster to user interactions, without sending to the server all actions and having to wait for the answer because it runs on the client. JavaScript follows the standard *ECMAScript [* 4], published by the European Computer Manufacturers Association (ECMA) and has evolved through different versions compatible with the above. In 2015, he published *ECMAScript 6*

(ES6) which provides better support for complex applications, allows creation and use of libraries *ECMAScript* compile target for other languages. Improvements in this new version of the standard are related to the modules *import* Y *export (* that allow static analyzers to build a complete tree of dependencies without executing code), class declarations, iterators, generators and asynchronous programming promises, among other things. SS6 also provides the basis for regular improvements and incremental language and libraries, which has meant that the standard *ECMAScript* into annual release cycles. The next release, in which it is working draft is currently the *ECMAScript 2019.* A the like languages such as Java or C, JavaScript values can be stored in variables, implement functions, operators use on standard objects or native types, etc. However, one of the most important features of this language is called Application Programming Interface (API). This type of operation interfaces based on inserting ready code to be reused in different applications, facilitating the work of developers. The API can classify into two categories:

and third-party browsers. The first includes those APIs that are built into the web browser and expose information about the device itself, such as the Document Object Model (DOM), geolocation, *Canvas* or *Web Graphics Library (* WebGL). For its part, the third-party API would be for example Twitter [5], giving *tweets* of a user and display them on a web itself, or Google Maps [6], with which you can embed custom maps among other features. It is, as already mentioned, a language with a strong presence on websites today, it is not compiled to run, but is interpreted and supported by all current browsers (all logic in the client). Use prototypes for inheritance instead of class (important difference with Java) and is oriented to objects and events.

2.1.2. React

This is a JavaScript library [7] open source, developed by Facebook and launched in 2013, whose goal is to provide developers creating interactive user interfaces. Currently applications often reach a level of complexity that requires the structure that are based is well organized, be flexible to adapt to changes and can be reusable in whole or in parts. This idea of dividing applications into modules is encouraging developers to use microservices based architectures, components or modules. Following this trend, Facebook library was precisely designed to be based on the use of components, which have a state that depends what they stand at all times. *props)* and its internal state, returned HTML with corresponding content. Are easy to test using test unit, they are declared with JSX [8] and must be written to SS6. It is also important to know the life cycle of these elements when developing applications React, because it depends on the behavior thereof is in the order and with the expected result.

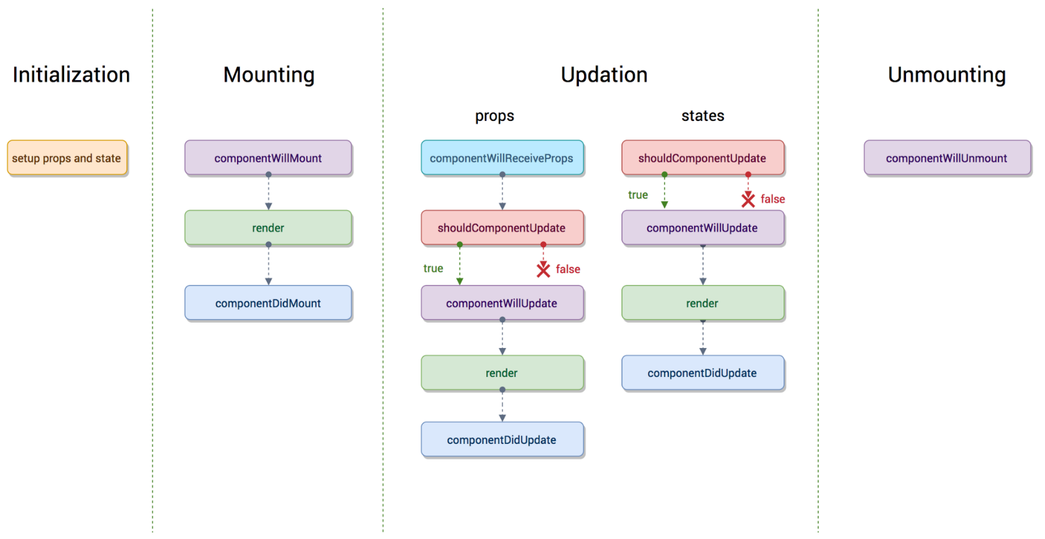


Figure 1: Life cycle of components React

There are more optional methods that can be implemented apart from those shown in the previous fi gure, but the *render* It is mandatory and must return a single element (either a DOM component or one React). Another feature of REACT is using a unidirectional data flow in the pattern Model View Controller (MVC). Thus, the events up to the most complex components and controls application status, while data down to the simplest components.

An important term within React library is the virtual DOM. This is a copy of the DOM of the browser that stores React and is used to determine which parts of it have been modi fi ed by comparing the version that is rendered with the previous one. Thus, ef fi ciently updated the DOM of the browser.

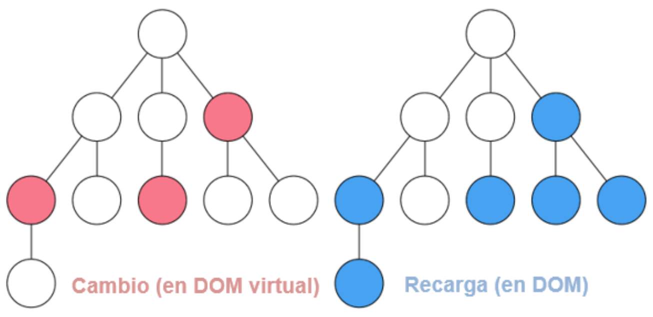


Figure 2: concept of virtual DOM

2.1.3. Redux

Redux [9] is a library that implements the design pattern Flux [10], although with some variations. It's a predictable container status JavaScript applications and helps write applications consistent and easy to prove. Before going further into detail about how it works Redux, it should be noted that the pattern or architecture Flux used Facebook for their applications and, therefore, is fully compatible with the components of React, as it also promotes dataflow unidirectional . Redux can be described in three fundamental principles:

- Single source of truth: the state of the entire application is stored in a single repository tree saved.
- The state is read only: the only way to modify the state is issuing an action (they describe what happened).
- Changes are made with pure functions are used *Reducers* pure to specify how the actions transform the status of the application.

The basics of Redux library are: the state stocks,
Reducers and warehouse. The state is immutable (modi fi ed not directly) and models the status of the application as a single JavaScript object. Actions are the only means which can modify the state. JavaScript are also objects and describe the change to be made. Actions are called because they are invoked due to interactions of the user with the application. The *Reducers* are pure functions (only use input parameters, do not resort to anything outside them) that apply actions on the state. They take as parameters the previous state and an action and return the duly modi fi ed state. The store brings together state and *Reducers*

to allow the latter to access and update the state records *listener* and manages the cancellation of this register via the function return *subscribe (listener).*

Redux for use in an application must be transferred React status and actions that can modified. For this, a component is created < *ReduxProvider />* state happens to the root component of the application. You can leave part of the state React components in the event concerned only relevant information for views.

2.1.4. webpack

Webpack [11] is a packager static modules for applications JavaScript. In the process the application, the tool creates a graph fi co dependency mapping each module necessary for the project and generates one or more packets with the processed application.

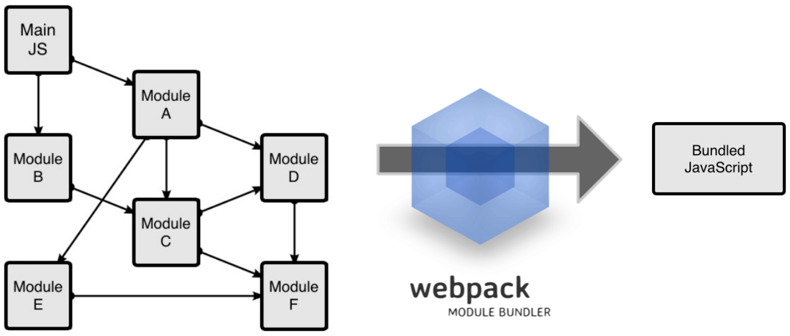


Figure 3: Packaging Webpack

One of the advantages of Webpack is the fragmentation of code. Only the necessary parts are loaded on each request, because not all components of a web application require full JavaScript code. Webpack is compatible with all current browsers (ES5 supporting the minimum) and provides the compiled code in other languages to native JavaScript, this compression or standards transpilación to ES5 *ECMAScript* more advanced, that they are not yet so widespread among browsers. During the development process server to package all the code and its dependencies, transpila automatically modi fi ed fi les and the application has changed in the browser is launched. To move the project into production, the tool transforms all the code and dependencies into a single *script (bundle).* Until version 4, they were strictly necessary one or more fi le con fi guration to package all the code. Thus they kept separate con fi gurations for the development and production of the project. Currently they are no longer needed these fi les, as Webpack has a default conguration. However, it is still widely fi configurable to suit the needs of each application, and for that the user can create a fi le called

webpack.con fi g.js in the root directory of the project that the tool will detect and automatically use. The fundamental concepts of con fi guration are:

- *Entry point:* It indicates the module which Webpack should start building the chart co internal dependency. The default is *./ src / index.js*
- *Output:* It indicates where the packages must be created and how to name files. By default it *./ dist / main.js (* main output file) and other files
  . / dist
- *Loaders:* They are those that allow Webpack process files that are not JavaScript and convert them into modules that can be consumed by the application and added to the graph fi co dependency.

- *plugins:* are used to extend the capabilities of Webpack, allowing you to perform tasks such as packet optimization, asset management resources (images, audio, video, etc.) or injection of environment variables.

2.1.5. React Native

React Native [12] is a *framework* which allows developing native applications using JavaScript and iOS Android and React. It works the same way as React, using the components that form a properly structured modern user interface for applications of these devices. React to the like, it has been developed by Facebook and its applications used in addition to continuing with the philosophy of open source. Native applications are those that are distributed to users through binary files and must be installed on devices (not consumed in the browser). React with Native can interact directly with the native modules mobile device, so you can create applications exclusively for Android, iOS or to work on both platforms. Instead of using HTML elements as is done in React, React Native in working with native components. Thus, elements such as < *div>, <header>* or < *article>* They are equivalent to < *View>* in React Native, and < *span> <p>* or < *h1>* equivalent to < *Text>.*

React Native provides various APIs to access native functionality, such as *Alert, AppState, CameraRoll, geolocation* Y *StyleSheet.* the basics are maintained and status React *props,* JSX and use of components, virtual DOM, etc.

2.2. Virtual reality

It is difficult to determine the exact origin of virtual reality (VR), because even though since mid-last century began trying in efficiency ction, has not been up to these last you decades that have developed devices that bring this concept to the market and, in recent years, increased commercial production due to the exponential curve technological development they have had. *Virtual reality is a computer technology that generates three-dimensional environments with which the subject interacts in real time, thereby producing a sense of immersion to such a presence in the real world [* 13]. There is an important aspect when talking about this technology and the concept of degree of immersion. Traditional video games in recent years that, even if the user interacts with a world fi ctitious and created by software, not feel completely immersed in it, fall into the category of semi-immersive virtual reality. There are different viewers to consume RV applications. From a simple cardboard structure with special lenses (Google Cardboard [14]) that can be used with any *smartphone,* to more comprehensive viewers that increase the degree of immersion of the user in the virtual world. This way you can interact with objects through controls or even recognize the movement of the user. Among these devices are the Oculus Rift [15] and Samsung Gear VR [16]. You can also view content 360 in browsers without using a viewer. Thus, it can consume virtual reality from different devices, varying the degree of immersion achieved. A term that causes confusion when being de fi ned and compared with VR is augmented reality. This kind of reality is based on the combination of the real world with the virtual, via software, to enrich fi c and interactive elements graphic visual experience.

2.3. Virtual Reality libraries

2.3.1. Web GL. web VR

Web Graphics Library (WebGL) [17] is a fi speci cation standard that defines a JavaScript API for rendering fi graph cos three dimensional (3D) in the browser, where the Processing Graph co (GPU) supported by the device. WebGL allows hardware acceleration and GPU processing images and effects as part of the *canvas* the web. Linked to this it is also the library of *Web Virtual Reality (* WebVR) [18]. In this case, we speak of an open fi cation speci makes it possible to run virtual reality experiences in the browser. WebVR allows WebGL scenes are presented on the screens of the devices as virtual reality scenes. These scenes are characterized by being consumed by a viewer (*headset)* getting through lenses specific characteristics, present the content to the user and with immersive depth.

2.3.2. Three.js

In the history of the development software you can find a huge variety of *frameworks* which aims to abstract the complexity of the API lower or more primitive levels, and so facilitate the work fi cient and developers. Three.js library [19] arises in order to simplify access to features WebGL and WebVR. It is a library written serves to create graphs 3D animated web browsers and can be used on WebGL JavaScript.

2.3.3. React VR

In April 2017, Facebook launched React VR [20]. This new *framework* open source sought to promote the creation of immersive content on the Web and provide virtual content developers added to your applications. During the year following its launch, the VR React community has used the resources of this

framework to create experiences as 360 rides or brand promotion, films and products through immersive environments. React VR follows the design React with a component-based structure. The main component is loaded from the fi le *index.vr.js,* where the application for the packager records also integrated on the home page (*index.html).* As React React and Native, applications have a state and a *props* They are descending from the father to the children components (fl ow unidirectional data) so that they take care of the rendering. At the time of launch there were two different versions of React VR: the open source version focused on developing for the web and internal version of Oculus [21] for mobile applications and computers (PC). Oculus VR is a pioneer in developing technology virtual reality, which was bought by Facebook in 2014. It is responsible for two of the best-known market viewers: Oculus Oculus Rift and Go [22]. Although both versions projects started from a shared API, they have since diverged to address different needs.

2.3.4. React 360

React 360 [23] was presented during the annual conference of Facebook, which was held in early May this year. This version, although it is based on the same principles and has the same objectives as its predecessor (React VR), focuses on improving aspects related content dimensional (2D). One of the main reasons for this decision was that this kind of content are where most of the work focuses developers who want to use *framework*

virtual reality to enrich their websites. Thus, in the private version are tools focused on 3D graphical cos fi.

React 360 has been designed to reach all users, even those without devices that consume the content virtual reality fully immersive way. The aim is to provide a better user experience on the web and can consume in a more interactive way and in an environment 360 content. highlights major improvements regarding React React 360 VR in its documentation or fi cial:

- Easier for creating 2D interfaces in 3D space. The introduction of the concept of surfaces (*Surfaces)* It facilitates positioning of both cylindrical and flat panels in the application. They can work with pixels rather than in meters.

- Improved support means. There are new features to improve media environment immersion. The developer has more precise control over the appearance of the application at all times.

- Better performance. It has been completely rebuilt runtime. The objective of the current architecture is to allow lower-end devices can also enjoy the RV content.

React a 360 application consists of the application code React and runtime (*runtime),* code that transforms 3D components React elements on the screen. Web browsers have a single thread of execution (*single-threaded),* so if the application crashes, rendering stop. In applications where consumption is expected to be on viewers RV, these blocks could break the sense of immersion and so in 360 React application code runs in a separate context (using a *Web Worker Executor* in current browsers, and *iframe Executor* in environments oldest). React when the code creates new elements, it indicates to the runtime, as when the user generates an entry with its interaction reaches React as an event. React components used 360 states and *props* as an input for the children rendericen components according to the state of the parent. React packaging in 360 projects as Native React, is using Metro, a packager JavaScript that works very similar to Webpack. The fact concerned a *framework*

As recently it affects the amount of features already implemented which may be provided, as examples, and developments of solutions found by the community of which it has reference. In React VR for example it has several components, and React Native are currently more than 50 components and various API. However React 360 has few documented components on your page or fi cial, among which we can find several legacy of Native React.

2.4. Communication between the browser settings

When JavaScript code is written for a website, usually necessary to use different Web API to access objects as *Body, BufferSource, CanvasPatterns, DOMCon fi guration, DOMObject,* or *EventListener* among others. One of the most common is *window [* 24] which it has methods, properties and events, and among them is the method *postMessage () [* 25], which provides a secure means for a send data to another window that is not necessarily in the same domain, as normally *scripts* different pages can only be accessed between them if the pages share the same protocol, port number and an trione machine fi (*host).*

With *window.postMessage ()* can be resolved so sure that restriction. To do this, you must get a referral from the window you want to send a message (usually a *pop-up* or *iframe)* and then use the method to send the message on an event object that receiver will be free to handle.

targetWindow.postMessage (message, targetOrigin, [transfer]);

- *targetWindow:* is the reference to the receiving window.
- *message:* the data are sent. Is serialized using the structured cloning algorithm, so you can send many types of objects without the developer must serialize previously.
- *targetOrigin:* specifies the origin of the receiving window to the event can be launched (scheme, *hostname* and port are indicated here must match those of the destination). It can be a literal ** string,* that would allow any origin, or be an identifier Uniform Resource Locator (URI).
- *transfer:* It is optional and is used to send objects transferred to the destination.

Chapter 3

Context

3.1. The authoring tool Ediphy

Ediphy [26] [27] is an open source web tool and resource authoring *e-Learning* which lets you create multimedia educational materials with a wide variety of content. This tool is developed in the Internet Group New Generation Engineering Department of Telematic Systems ETSIT-UPM. To the being a web platform, it is not necessary to install any additional software to use and can be consumed from any device with a browser installed. It is a project developed in Javascript (SS6), along with libraries like React and Redux. Ediphy has a modular architecture based on components called *plugins,* allowing it to be extensible and facilitate increased functionality in a simple way for programmers. The tool can be used in editor mode, where courses are created, or viewer mode, where they are consumed.

3.2. Plugins API and plugins creation

As it indicated in the documentation or fi cial tool, the smallest unit of content which is based Ediphy *plugin,* concept that refers to the representation of an editable box and / or fi gurable with some kind of substance, as it could be a text. there are several *plugins* catering to the needs that arise in creating an educational resource, and the tool interface are grouped into categories according to their nature. The *plugins* They can be added to both documents as slides (the two possible types of view for courses that are created with this tool) and some have extended their functionality (enriched) and allow the creation of hot spots (marks) to navigate between views and external content, generate a content linked to the *plugin,* or throw explanatory text as *pop-up.* Ediphy currently has numerous *plugins* that help users have a wide variety of features for creating educational resources. Among them we can highlight images, graphs, video players or webs embedded.

Because of the open source philosophy (*open source)* which follows the project may contribute to creating a *plugin* itself, in which components can be reused existing React. To do this, the tool has an API for creating
plugins which allows developers from a base set minimum requirements required for the implementation of new functionalities. With the execution of the command *yarn run create-plugin name plugin* a new folder is created within the directory *plugins* including all fi les needed to start developing. The API is also prepared to specify input the category to which belongs *plugin,* if you enriched or need display template.

```
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) {}, auxiliares_ _funciones: function ( event, element, parent) {}}}
```

In the skeleton of the *plugin* for the editor shown in the previous figure fi, you can see the base object receives as a parameter. This refers to the *plugin* the basis upon which all the *plugins* of the application. You need to use the access methods and modi fi cation of the state, and to record auxiliary functions so they can be used by any other. most important, being mandatory for all functions *plugins* regardless of the complexity or

requirements from development, They are *Fig getCon ()* Y
getRenderTemplate (state). The first returns a single object with the configuration of the
plugin. Some data gurable fi are the name, the category to which it belongs, the icon that appears next to the name, the way it is written (JavaScript pure or REACT) or the initial values of the appearance of the box in which it presents the *plugin.* The second mandatory feature aims to return a text string, either through HTML elements or components React but always returning a single root element other than a label resealable.

Chapter 4

Developing

4.1. Requirements capture

Prior to the selection of technologies and implementation it is necessary to capture the requirements that mark the lines of the project. In this case, as chosen tool allows the creation of educational content, the distinction between two types of requirements are necessary: educational and functional.

4.1.1. educational requirements

Since Ediphy is a tool for creating educational resources, and to determine the characteristics of the plugin that aims to add, they have taken into account the educational requirements that must be adjusted.

- Providing virtual reality environments that enable the user role student to learn in a different way and simulate immersion in environments that can extract learning

- Add the possibility of fragmenting the information content in small pills

- Facilitate the relationship of the information provided and the educational virtual environment to easily adjust to the educational fi n

A use case could be learn the planets and their most important features. Through a virtual walk through the space, you can click on any planet or star and get your information. It is a way to provide the user with role a form of playful student learning based on observation of a virtualized environment that simulates a planetary shift without leaving the classroom.

4.1.2. functional requirements

To develop the educational content enriched using virtual reality technology aims to provide users a more dynamic and immersive environment.

- One of the main requirements that must have application of these features is the ability to add components capable of providing contexts and scenarios that will generate depth and realism. Moreover, in this sense it will be necessary for the user to choose their own scenarios.

- The application must also allow background audio is added to increase the degree of immersion in scenarios 360. The sound environment is an element that contributes to the sense of immersion, allowing introduce the user within the virtual world not only through the sense of sight, but using hearing.

- Another functional requirement is the inclusion of objects that interact, allowing be placed at different distances to again increase the sense of immersion, and facilitate learning through the use of information in small pills, in principle textual.

- Finally, it is considered necessary to have an element carousel picture mode that allows way more complete graph the information given in the application.

In short, these are the requirements identi fi ed that will translate into features and allow us to build three-dimensional virtual environments rides:

- 360 scenarios with custom background
- Enable / disable Audio environment
- dimensional objects
- Buttons that interact
- 2D projection images

4.2. Technology selection

Ediphy is a complex tool with many features and evolving development. It React used as the basis for all its architecture and for this reason React VR, a library of the same group for creating virtual reality content, it was considered a good option. This library was a few months of life starting this work (April 2017), but in the context of Ediphy was presented as the best option to extend the functionality of the tool with the development of a *plugin* virtual reality. React React VR is based on Native, and both libraries share modules and basic API. However, React VR has different characteristics that give it potential to create virtual reality content on the web through speci fi components cos.

During the development work, particularly in early May this year, Facebook decided to leave the line of work that had followed his library of virtual reality and rename it in a new version that leads back to goal of enabling developers I create content virtual reality to consume in web browsers without knowing concepts natives of these technologies, but based on the use of JavaScript in order to create content more immersive, interactive and attractive to users. This new library is called React 360 and changes the operation of the main components and features of the flow of execution, though there are some similarities with its predecessor.

4.2.1. VR research React

React VR focuses on typical aspects of 3D applications, ie three-dimensional objects take precedence intended to be rendered in space, according to a coordinate system based on meters. A React being a library, the application is built using a component-based architecture, which have a single standard procedure: *render ().* Since this method are rendered on the screen the components necessary to build a scene or part of it. This is a list of some of the components that account React VR:

- *Cloth:* controls the application background. Receives images in your attribute *source* and rendered in an environment 360.
- *VideoPano:* Pano like, but in this case used as source videos recorded by special cameras, 360.
- *Model:* 3D rendered objects created. Among its attributes also supports file materials (. *MTL)* corresponding if necessary.
- *VrButton:* It is responsible for wrapping other components to equip them with the typical interactive buttons, such as events *click,* focus on them or selection. They have no physical appearance alone, but must serve as packaging for other components as < *Text>, <Image>* or < *Model>.*
- *Image:* rendering 2D images into positions on the coordinate axis of the virtual world, ie, dimensions points.
- *Video:* videos 2D rendering positions on the coordinate axis of the virtual world, ie points with dimensions.
- *Sound:* It enables a sound source that emits the contents of the audio file that is passed to the source attribute in the component in which it is involved.
- *View:* As to the < *VrButton>,* this component does not have its own physical appearance to render, but serves to wrap other components. You can be found in HTML equivalence with the < *div>.*
- *Text:* 2D text rendering in positions on the coordinate axis of the virtual world, ie points with three dimensions.

React VR in both 3D rendering elements on the screen of the browser as the application logic running on a single thread (browsers have one main thread execution). This fact causes any delay that causes the application code, will increase the latency rendering of three-dimensional scene. This is a critical aspect when consuming the contents of virtual reality viewers, since the feeling of immersion is lost. React VR in the entire application is written from the le fi *index.vr.js,* where the main component is recorded and *Client.js* He is responsible for preparing for compilation and integration as JavaScript *index.html.*

4.2.2. First approach: VR React

During the first part of the preparation of this work, the current library was React VR and the proposed scenario focused on the use of 3D objects.

The teacher selects a fund to acclimate your plugin Ediphy virtual reality. To do this you can choose between providing the Uniform Resource Locator (URL, for its acronym in English) of an image resource or video 360 type 360, or select one of those who already have the default application. Enter by uploading files containing 3D objects, models in three-dimensional space, the coordinates and the scale you choose for each. You can choose whether to encourage them so that rotate about its own vertical axis or not, giving the possibility to offer more dynamism to the scene. Professor, when editing the component, you can also enter resources such as videos or images to be projected at a certain distance from the user, In addition to enabling the same basic controls (play, stop, next and previous). The student consumes the generated content by the teacher so that you can navigate the world of virtual reality, observing models or graphic resources speci fi c have been added, and controlling the latter in the event that are enabled controls for it.

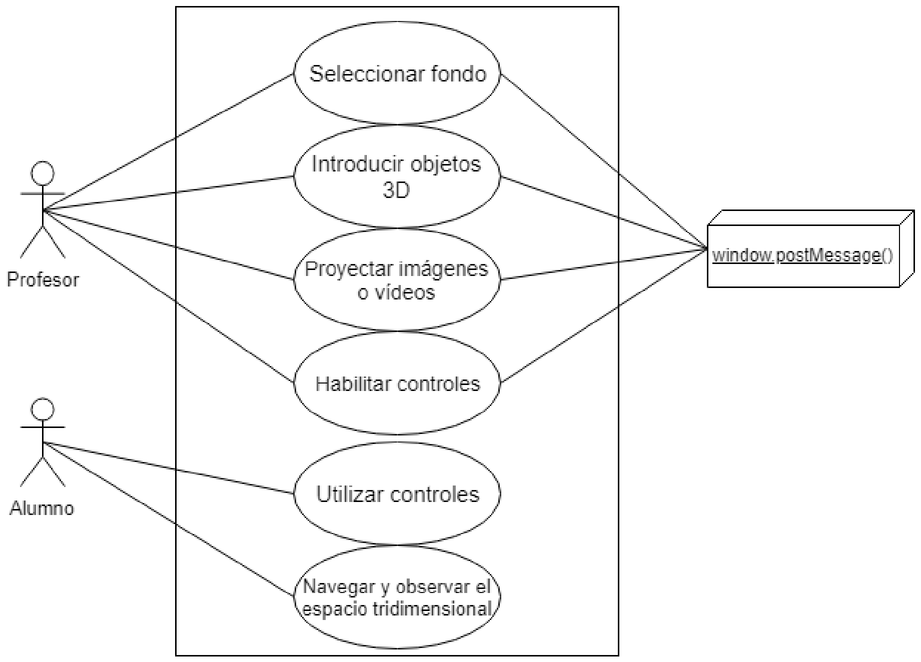


Figure 4: Scenario proposed with VR React

4.2. SELECTION OF TECHNOLOGIES

Two tests were made using the concept described elements on stage to check the suitability of the result before you start implementing an application that would allow editing from Ediphy. The first was based on several examples that developers have been published during the year has been launched this library. animated buttons were tested in the form of text and images, functions to change the background image and a menu to choose between certain videos that were projected.

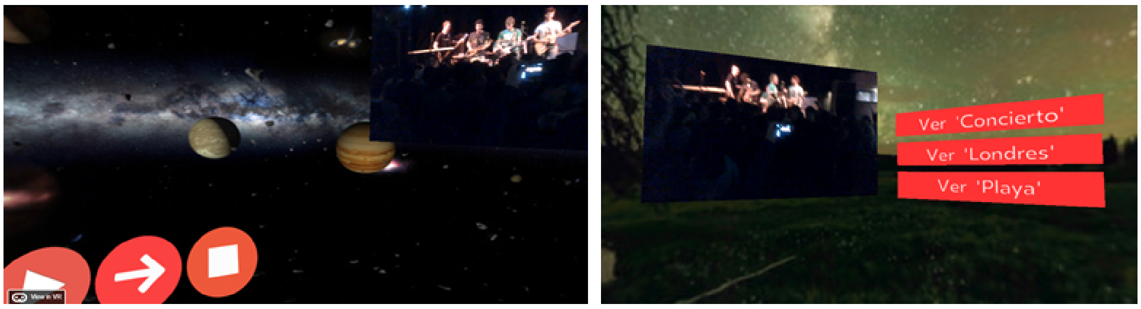


Figure 5: Proof of Concept with VR React

The main component of the application, which renders *index.vr.js,* It has a state from which the exchange line (manual or automatic) and the video projected controlled. The remaining components are built based on the most basic offered by the library, such as < *View>* to group, < *VrButton>*

to provide images and text menus animation and interactivity, < *Pano>*
y < *VideoPano>* for image or background video, < *Sound>* for sound environment,
<Image> y < *video>* audiovisual resources for that project and < *Text>* for the content of the menu items. The second test was to replicate one of the best known examples of this library on the Internet [28]. This is proof of concept that finally failed to be used in the fi nal application of virtual reality to Ediphy by changing library.

4.2.3. React Research 360

React 360 comes to break the trend that marked the three-dimensional approach React VR, and focuses on improving rendering 2D content within a virtual world 360, adapting to the line of work they pursue from Facebook with this version open source, which is based on the animation and improved interactivity of web content. One of the biggest differences between the two libraries is to use what is called runtime. It is separate the context of the implementation of rendering, ie conversion code React to 3D items on the screen. Thus, it allows the cycle representation is constantly upgrade to a high frame rate as the React code creates new elements, tells the runtime that add to the 3D scene. Conversely, if the user provides an input, is returned to as an event React. In this case, the fi le in hosting the main application code it is *index.js,* in which the other components hanging from east to form part of the application is imported. The runtime code is in the fi le *Client.js,* which connects the application with the browser. In this instance a new fi le React 360, is loaded creates React code and connected to a speci fi c place in the DOM (provided by *index.html),* in addition to the initialization options which specifies that the developer. Subsequently, this file is responsible for rendering the code React on the scene, attaching the component declared *index.js* the predetermined surface and other components (if any) to the surfaces or locations indicated. Optionally, you can add a background image to display 360 while loading the code React, improving the perception of the user.

The other major difference in which 360 React React improves the characteristics of VR is the inclusion of the module surface (*Surface).* With it, it is easy to create 2D interfaces in 3D space and can work in pixels rather than in meters. This way, you can use specifications created with traditional design tools and code sections React Native. Some of the components that are inherited from account React VR (*View, Text, Image, VrButton).* But nevertheless, *Model* has been renamed *Entity* and now supports object formats. *obj* Y . *gltf,* with corresponding fi le materials (. *MTL).* In addition, React 360 has several API:

- *animated:* It was originally implemented to React Native and now is the same that is used for virtual reality library React. With it you get alternate the size, position and appearance of the elements in the scene so that effects are generated to help us enrich the interactivity of the user with these elements.

- *AsyncStorage:* It is a simple JavaScript API keyvalue storage that preserves data between multiple application loads in the browser.

- *ControllerInfo:* It provides information on external devices such as knobs and controllers connected to the computer. It is used to respond to events connection and disconnection of the controller and can extract information about the drivers static (unique identi fi ers, counts buttons and axes, left to right, etc.).

- *Environment:* It is used to change the appearance of the background of the scene, either by using static images with formats to render in an environment 360 (mono / stereo equirectangular), or by using videos in these formats. It replaces the functionality that had *Cloth* Y *VideoPano* in ReactVR.

Using the initial examples that projects these libraries are loaded, you can see the difference in approaches has commented on the intended use from Facebook developers make their library.

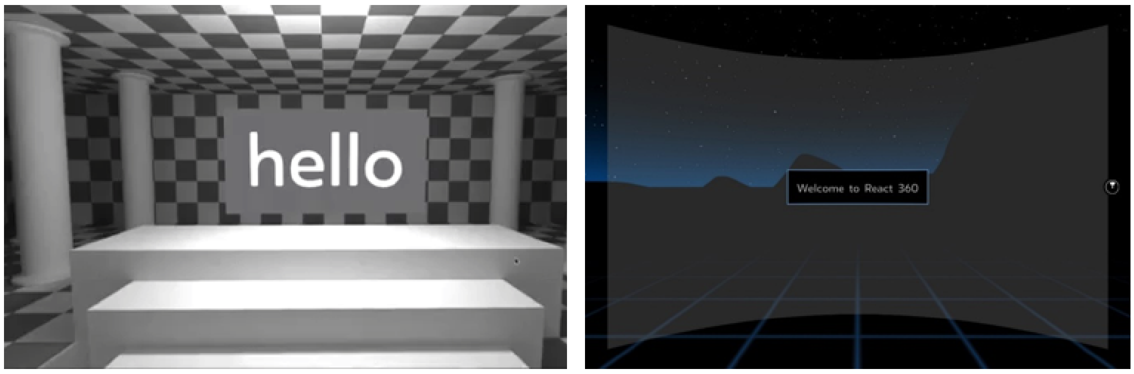


Figure 6: Comparison of initial projects

4.2.4. The definitive scenario: React 360

After changing library, the proposal of this work had to be re-evaluated based on the new features offered in React 360. On the website o cial fi indicated to facilitate the transition from one to the other components React VR they are still available in the new package. However, this may change in the future as it has been restructured operation and the way in which some components are rendered, so you may end up getting obsolete.

The teacher can select a background to acclimate your plugin Ediphy virtual reality. To do this you can choose between providing the URL of an image 360, *or select some of which already have the default application. Enter through plugins enriched tool for Ediphy, marks in three dimensional space in the coordinates for each point. These marks may be emerging type (show the selected text to be) or link to another slide or page of the course you are editing. You can choose whether to enable the basic controls of the audio environment (default audio). The teacher can also select up to ten images to be projected at a certain distance from the user. The student consumes the generated content by the teacher so that you can navigate the world of virtual reality, observing the marked points and images of the projector and control playback or pause the audio environment,*

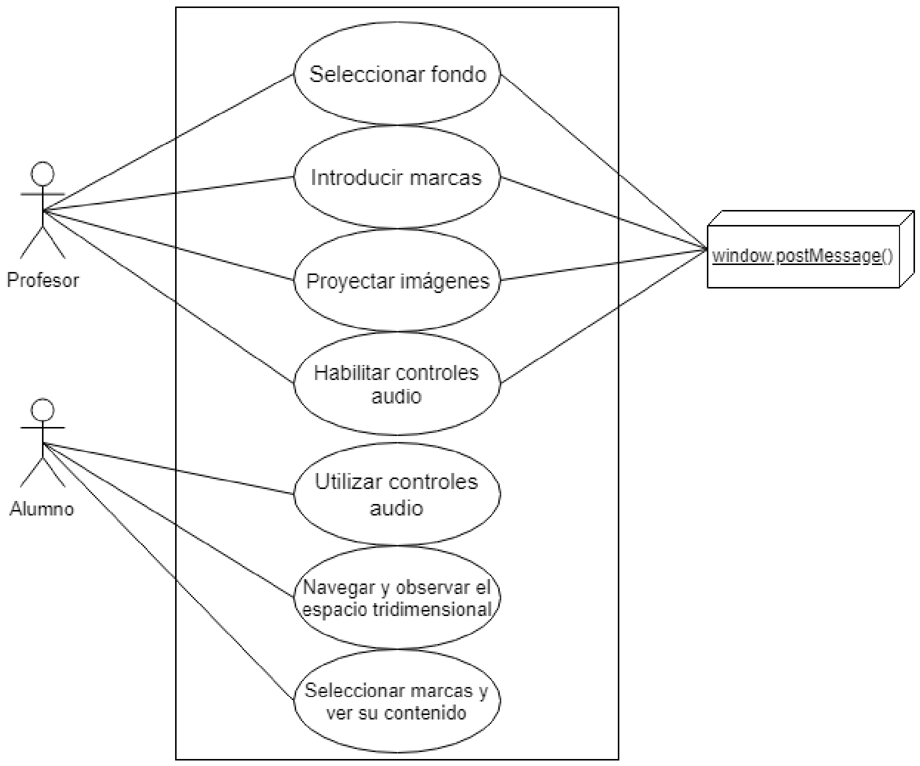


Figure 7: Scenario proposed with React 360

No evidence for this scenario was not realized concept, since that would change the basis of library work in the middle of the development caused the rede fi ne need for the objective of the *plugin,* again studying the documentation and the possibilities of the new library, so that development should be as efficient as possible in terms of time spent on each step in implementing the fi nal application.

4.3. Implementation

4.3.1. React integration of 360 in Ediphy

As shown in the documentation or fi cial, there are two ways to integrate virtual reality application on an existing web site using *iframe* or runtime adding a JavaScript wider application. However, the second option is not recommended because of the need for the application code being packaged beforehand, This requires adding certain packager extra functionality to use the application at first, besides representing an extra burden of several hundred Mega Bytes. Therefore, to integrate this application in Ediphy decided to follow the recommendation or fi cial and use the path of the *iframe.* By attribute (*src)* HTML tag < *iframe>,* It points to the file *index.html* the application of virtual reality. In this way, no inter refers to the main web page and so this does not slow down rendering application.

At the same all libraries React during the development of the application code is compiled and dynamically packaged by a local server that launches the application and is available at a port of the machine (ie, default is 8081). On the other hand, when the implementation of the code of the application is completed, the project goes into production running one of the *scripts* React default schedule 360. In production, all the code necessary to run the application is packaged in a single directory, which contains compiling JavaScript code production and a copy of the file *index.html* pointing to JavaScript compilation. If resources are used staying static file folder in the application (*static_assets),* it must also be copied into the production directory on the server where it will stay the application. For this project, once completed development, production directory folder takes *dist* in Ediphy and amending the absolute address pointing to the attribute *src* of the *iframe,* a route on the le fi *index.html* of production. After determining how to integrate the application in Ediphy React 360, the next objective was to communicate. The question was: £ how to make two different windows (one *iframe* embedded in a web page) can send messages to each other? The element itself windows in browsers (*window)* It has a method whose description indicates that its purpose is precisely to allow safe way communication between windows with different backgrounds, whether independent web pages a *iframe* within a page or pop-up window. It is the method *postMessage () (* explanation of this method may be detailed in the State of the Art). By using this feature, we have designed a messaging protocol for connecting and sending information between *iframe* React 360 loading the web application and Ediphy. This information contains the data for the application state and with them the components thereof are rendered accordingly. The following diagram shows an example of user interaction with the Ediphy application, using the *plugin* virtual reality to put a mark in a place which has rotated by moving the camera. Then in preview, will select the brand.

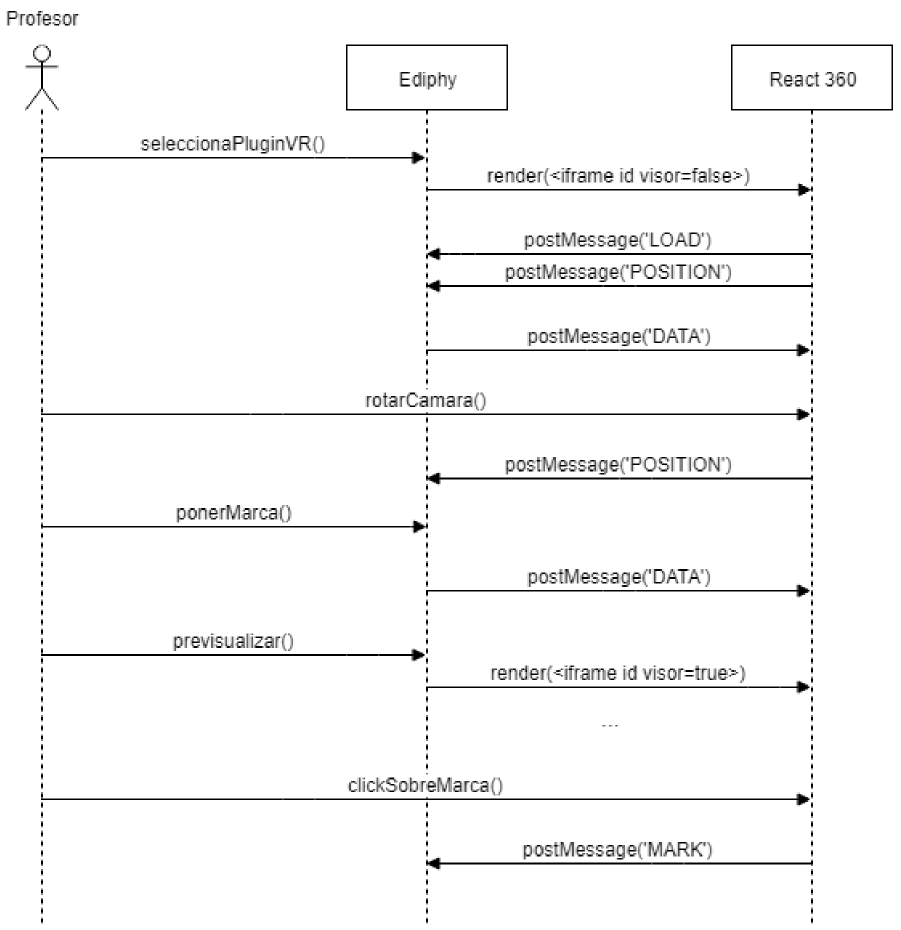


Figure 8: Protocol messages *postMessage ()*

As shown in the figure, Ediphy reacts to messages received according to the parameters. The connection between the two applications the virtual reality begins with the message LOAD. This decision was made after testing sending messages from Ediphy with data from the state. In trying to access the preview window (which renders a new *iframe),* the application was not loaded because the state did not receive. The result was that in the viewfinder display is not getting the changes made in the editor, but the application was 360 in the initial state. This ruling was solved with the new order in the protocol whose messages indicate the full charge of the application to Ediphy can send e fi ciently data (DATA message) on the status of the *plugin* edited. To send them, I had to check that the attribute identifier (*id)* window sending the message LOAD corresponds to the *id* which has kept Ediphy for that *iframe*

concrete.

```
let query = decodeURIComponent (window.location.search); try {

    let myurl = query.split ( "? Id =" );
    var sinNomId = myurl [1];
    var sinNomVisor = sinNomId.split ( "& Display =" );
    var id = sinNomVisor [0]; IDBox = id;

    window.parent.postMessage (JSON.stringify ({msg: 'LOAD' , Yo hice}), "*" )
```

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

```
toolbarUpdateValue (props = Este . props) {
        let receiverWindow = Este . 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}, "*" );
}}
```

The application to the load also sends a message POSITION with the coordinates of the current position that is being watched (rotation). This data is checked every 2 seconds and compared with the last calculated so that if you have changed the position to which the user looks, a new message is sent to Ediphy POSITION.

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

```
handlePosition (position) {
              window.parent.postMessage (JSON.stringify ({msg: "POSITION" , Id: IDBox, position}), "*" )}
```

```
if ( Este . windowSource && data.msg === 'POSITION' && data.id === Este . props.id) { Este
        .  SetState ({position: data.position})}
```

Furthermore, there is another type of message to indicate from React application 360 has been pressed on a mark (message MARK) whose functionality to navigate through the resource Ediphy or go to an external content. To do this, he indicated the tool *iframe* where is the brand and its data.

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

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

This second PostMessage function parameter indicates what should be the origin of the target window for the event is sent. If the scheme, the host name or port number does not coincide with those of the document window target, the event is not sent. In the previous code snippets you can see how this parameter is sent with value *. Thus not checked the destination to which data is sent.

4.3.2. React 360 Application: Connection module

One of the fundamental differences between VR and React React 360 is using runtime to improve application performance. This separation

contexts that since the application React can not access certain web API methods (as are those who need our project: *window.postMessage ()* Y
window.addEventListener ()) the browser is available only for the main thread. The solution offered by developers of the library is to use native modules. Some of the most common modules are available in 360 React package for web, as *Location* Y *History.*

Native modules extend the class *module* and must be created and recorded during startup of the application (in *Client.js).* They must be imported into the components that will use them.

```
import ConexionModule from './ConexionModule' ;
function init (bundle, parent, options = {}) {const r360 = new ReactInstance (bundle, parent, {

    fullScreen: true ,
    nativeModules: [ctx => new ConexionModule (ctx)], ... options,});
```

For the application that is developed in this work has implemented a native module called *ConexionModule,* whose purpose is to receive and send messages using *postMessage ()* and determine, according to the parameters that you call the application, what is your identifier in Ediphy and if it is or not being shown in the display. This element code begins by importing the class *module* of the package

REACT-360-web, since the native module that is believed to extend this class. In the constructor is called the first to the class to which extends passing as parameter the name of the native module is created, so that it is accessible in the component *NativeModules* React 360 *NativeModules.ConexionModule.* the context and data source window that will be used to connect to the window in which the application is initialized via insert *iframe.*

```
import { Module} from 'REACT-360-web' ;
let IDBox = null ;
export default class Module extends ConexionModule {
    builder (ctx) {
        super( 'ConexionModule' ); Makes // esta module available at NativeModules.
                MyModule
        Este ._ rnctx = ctx;
        Este . Winsource = undefined;
        Este . origin = undefined;
        Este . handleMark = Este . handleMark.bind ( Este );
    }
```

Then the functions implemented this module detailing, which are available for use by the application code to be imported from
NativeModules (and previously registered *Client.js).*

- *conexionIframe (callback):* This function has two parts and is called before rendering the main components, which need to send some information Ediphy them to be painted correctly. The first part is independent of any event but directly collected from the *query* parameter identification and sends a message LOAD to the window in which it is embedded application. In the second, it is expected for a promise that will collect the data you send when Ediphy has validated the connection through a listener of events enabled for it. In addition, stored in the variables that were discussed in the constructor so that they are accessible to all other functions, without waiting to receive any event which draw information from the destination.

```
conexionIframe (callback) {
     result = const new Promise ((RESOLVE, reject) => {
        window.addEventListener ( "Message" , function ( event) {
           Este . Winsource = Event.Source;
           Este . origin = event.origin;
           if ( event.data.audioBack) {
              Este . winSource.postMessage (JSON.stringify ({msg: "State audio
                    received correctly" }), Este . origin);}
           if ( event.data.urlBack) {
               Este . winSource.postMessage (JSON.stringify ({msg: "Back received Url
                      correctly" }), Este . origin);}
           if ( event.data.imagenBack) {
              Este . winSource.postMessage (JSON.stringify ({msg: "Image Back
                    received correctly " }), Este . origin);}
           if ( event.data.imgs) {
              Este . winSource.postMessage (JSON.stringify ({msg: "Received Images
                     correctly" }), Este . origin);}
           if ( event.data.showPanel) {
              Este . winSource.postMessage (JSON.stringify ({msg: "Show Panel
                    received correctly" }), Este . origin);}
           if ( event.data.marks) {
              Este . winSource.postMessage (JSON.stringify ({msg: "Marks received
                    correctly" }), Este . origin);} resolve (event.data); }), function and () {console.log ( "Problems with
          the promise" );};

     });
     result.then (data => {
        if ( Este ._ rnctx) { Este ._ rnctx.invokeCallback (cb [data]);}});
```

the Function ON *callback* It is responsible for collecting the necessary data for each component and update their status as needed. For instance, in the component control marks (*Mark),* This function is invoked from the *componentDidMount ()* and the result status is updated with the brands Ediphy have registered for this application (identi fi ed by the id field).

```
     escucharConexion () {
       ConexionModule.conexionIframe (data => {
         if ( datos.marks) { Este . SetState ({marks: datos.marks})}
         Este . escucharConexion (); })}
```

In the data projector used to store the address JSON resource that has decided to place on the panel.

```
     escucharConexion () {
       ConexionModule.conexionIframe (data => {
         if ( datos.imgs && datos.imgs.length> 0) {
           Este . SetState ({
                 arrayImgs: datos.imgs, keySelected: 0,

                 currentImg: datos.imgs [0] .currentImg,}); }

         if ( datos.showPanel) {
               Este . SetState ({
                 showPanel: datos.showPanel.show,}); }

         Este . escucharConexion (); })}
```

Finally, the component *index.js* It is used for the configuration of the environment:

```
   escucharConexion () {
       ConexionModule.conexionIframe (data => {
         if ( datos.audioBack) {
           /*try{AsyncStorage.setItem('showAudio 'datos.audioBack.play);} catch (
                 error){}*/
               Este . SetState ({
                 showAudio: datos.audioBack.play})}

         if ( datos.urlBack) {
           /*try{AsyncStorage.setItem('urlBack 'datos.urlBack);} catch (error) {} * /
           Este . SetState ({
               urlBack: datos.urlBack})}

         if ( datos.imagenBack) {
           /*try{AsyncStorage.setItem('imgBack 'datos.imagenBack);} catch (error)
                 {} * /
           Este . SetState ({
               imgBack: datos.imagenBack})} Este . escucharConexion
       ();}); }
```

As can be seen in the comments of the code, it has been tested using the JavaScript API *AsyncStorage,* used to store data in the browser status. This functionality was left commented because it was not needed for the creation and application development.

- *Envisor (callback):* In the same way that the identifier of the box is removed from the *plugin,* You can also removed the Boolean value indicating whether the *iframe* in which the application is being consumed is in viewfinder mode or editor.

```
Envisor (callback) {
        let query = decodeURIComponent (window.location.search); try {

            let myurl = query.split ( "? Id =" );
            var sinNomId = myurl [1];
            var sinNomVisor = sinNomId.split ( "& Display =" );
            var display = sinNomVisor [1];
            if ( Este ._ rnctx) { Este ._ rnctx.invokeCallback (cb [display]);}} catch ( and){}

}
```

This information is used by both brands and the role that plays the audio environment to disable the content that should only be consumed from the viewer (following the policy of Ediphy).

```
  handleMarkClick () {
        switch ( Este . props.connectMode) {
            case 'Popup' :
            let = ConexionModule.enVisor viewer (viewer => {
                if ( === viewer 'True' ) { Este . setstate ({show: Este . state.show})}}); return ;

            default:
                Este . props.onClick ( Este . props.id, Este . props.origin)}}
```

In the event that the mark is not type emerging, but the associated action is linked to another content of the course, the check is not required since the component of the viewer has no way to interpret MARK messages or launch the method relevant (see section creating the *plugin* in Ediphy).

```
  _playAudio = () => {
    let = ConexionModule.enVisor viewer (viewer => {
        if ( === viewer 'True' ) {
            AudioModule.playEnvironmental ({source: asset ( 'Audio / Blue_Jacket.mp3' ), Volume: 0.7,})}});

  };
_stopAudio = () => {AudioModule.stopEnvironmental ();};
```

In the case of the function for audio is not required this information.

- *handleMark (mark, id):* As in the section that explains the structure of the *plugin* the function that invoked the action associated with a brand that is selected in this case what you have is the function that sends the necessary data from the application of virtual reality to Ediphy, so that it knows what to throw indicated. Ie sends the message MARK (I see communication section).

- *handlePosition (position):* As the above function, this case has also been mentioned previously in explaining the protocol messages exchanged between the application and Ediphy. This is the method that sends the message POSITION, invoked when there have been changes in the rotation since the last interval (see section I communication).

4.3.3. Virtual Reality Plugin Ediphy

It has used the command *yarn run create-plugin VirtualReality* for the creation of the *plugin* virtual reality developed in this work. As stated in the documentation or fi cial to add to Ediphy just put it on the list

plugins of the fi le / *core / con fi g.es6*.

Conguration of the *plugin* editor mode (*VirtualReality.js)*

Once the skeleton has the *plugin,* the two mandatory methods of this for the editor are *Fig getCon ()* Y *getRenderTemplate ().* In addition, the application to edit a toolbar and an initial state is required.

- *Fig getCon ():* It was used to give the name and category *plugin.* The name that is displayed to users (property *displayName)* is selected according to their value in the fi le Translation (Ediphy is available in English and Spanish) and pooled with *plugin* multimedia. He *plugin* React is written in code, you do not need to fi setup prior to the launch and must have dimensions suitable to enter the slide or document in which it is inserted. He *plugin* must fi gured as enriched (*isRich)* and define the type of marks (in this case, the brands to be in a three-dimensional scene must be 3 coordinates). The property

  needsPointerEventAllowed It is necessary to enable the user to choose between move the virtual reality scene or box containing the *plugin* and places it in the document.

```
getConfig: function () {
                    return {
                           yam: 'VirtualReality' , DisplayName: i18n.t ( 'VirtualReality.PluginName' ) Category: "multimedia" ,
                           Flavor: "REACT" , NeedsConfigModal: false ,



                           needsTextEdition: false ,
                           initialWidth: '450px' , InitialHeight: "car" , InitialWidthSlide: '60%
                           ' , InitialHeightSlide: 'fifty%' , Icon: 'Event_seat' ,
                           NeedsPointerEventsAllowed: true ,



                           isRich: true ,
                           marksType: [{name: i18n.t ( "HotspotImages.pos" ) Key: 'Value' , Format: '[X and Z]' , Default: '0,0,0' , DefaultColor: '# 17

                                          cfc8 ' }],
};}
```

- *getRenderTemplate ():* To have the methods that have all components React, a component was created to render the *plugin.* That is, from the very *plugin* I could not call methods like *componentDidMount ()*
or *componentWillReceiveProps (nextProps),* which they were necessary for the implementation, so that the component was unbundled from the *plugin.* As shown in the above code, this component are passed as attributes (*props)* the unique identifier of the *plugin,* the state and marks (empty array if no marks have been placed). The drop zone marks the *plugin*

it is enriched with elements gured fi for classy
dropableRichZone in Ediphy, so a block is added to the same dimensions as the virtual reality component, to allow complete freedom insertion marks.

```
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 ():* This function returns the initial state of the application. Has related information with state of the features of the 360 React application (we will talk about them later).

```
getInitialState: function () {
                return {
                        imagenBack: undefined, urlBack:
                        undefined, audioBack: false ,

                        showPanel: false ,
                        numberOfPictures: 1,
};}
```

  the function of each of the parameters of the state as needed for the features listed below will be explained.

- *getToolbar ():* From here they can be structured options toolbar. Each tab has a name and many accordions as necessary for the *plugin.* In turn, accordions have a key and a name and an icon related to the grouping tools. These tools are buttons, objects with properties such as name, type, options selection types, icon, value, etc. This function receives as parameter the state of the *plugin,* so that you can update whenever the value of any of its buttons changes. Specifically, for the application of virtual reality they take two accordions, conguration and background, in addition to which is added to being a *plugin* enriched to put brands.

React to the editor component (*VirtualRealityPluginEditor.js)*

Of the component should be highlighted two fundamental aspects: the charge of rendering the *iframe* by the application of React 360 in Ediphy integrates and manages connection messages to communicate both to send and receive status position. Component code shown below.

```
builder (props) {
        super (props);
        Este . state = {position: [0, 0, 0]};
        Este . toolbarUpdateValue = Este . toolbarUpdateValue.bind ( Este );
        Este . receiver = Este . receiver.bind ( Este );
}
render () {
        return (< iframe className = { 'VR' } Allow = "Vr" width = '100%' height = '100%'
                    x = {data- Este . state.position [0]} = {data-and Este . state.position [1]} = {data-z
                        Este . state.position [2] = {} src 'VR / index.html? Id =' + Este . props.id + "& Display = false" } Id = "Receiver"

                        />);
        }
        componentDidMount () {window.addEventListener ( "Message" , Este . receiver);} componentWillUnmount () {window.removeEventListener ( 'Message' , Este
        . receiver);}
```

The state only keeps the position sends the virtual reality application because the identifier, state of the *plugin* and the marks put already passed from the fi le of the *plugin* to the shaped component *props.* To hear the message arrival event and manage the component should add event listener (*eventListener)* before mounting point to the function that manages the connection, and remove it when you go to dismount. Once the connection is established between the *iframe* and Ediphy, the function that is responsible for sending the updates made in the toolbar is *toolbarUpdateValue (this.props).* To invoke the component needs to implement the method *componentWillReceiveProps (nextProps)* so that whenever you modified the state (you edit something in the toolbar) or brand is added, you call the function that sends the update.

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

To disable features such as play or stop the audio environment, or launch actions marks in *query* that accompanies the source fi le with linking the *iframe* It has been added to the "viewer" parameter indicating the application with a boolean value if these features should be enabled or not.

Conguration of the *plugin* Viewer mode (*VirtualReality.js)*

For the viewer, the only function that is implemented is to render the virtual reality component. This disaggregation of component and *plugin* in two different fi le is due to the same reason as in the editor.

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

Only changes from the component to the editor in step of the method
onMarkClicked (), they have all *plugins* marked as enriched and used to manage the actions launched brands being activated.

React component for the viewer (*VirtualRealityPlugin.js)*

The differences between the two components are minimal. As in the editor, the methods implemented in this fi le aim to render the
iframe and manage the connection. The display attribute in this case indicates that it is true, so the application of React 360 enables the sound control environment and launching actions speci fi ed in brands have been created. However, in this case the coordinates of the location are stored to which the user is watching and it is not necessary to know for any of the functions of the *plugin*

in the viewfinder. As it is seen in section communication, rather than POSITION in the viewer what is MARK managed to launch from Ediphy, knowing the brand data that the user has selected the appropriate action. The last difference with
the component editor It is not required that the method
componentWillReceiveProps (nextProps), as this only runs when the application changes because editing (by modifying the values of the toolbar). Ie, the display is only necessary to know the status of the application with the changes that have been edited, the identifier of the *plugin* to communicate properly and receiving messages MARK to launch the appropriate actions.

4.3.4. Functionality adjusted to the context Ediphy - React 360

This section code [29] has been implemented to develop each of the application features detailed. In all cases, both the share Ediphy (toolbar) as the application is explained.

Background image

One of the basic features of the *plugin* It is to change the background image of the scene. To do this, you must add items to the toolbar that allows to the user to choose the image you want. This information is sent to the application of virtual reality, which will indicate to the background component (*Backgroud.js)* the image to be rendered.

VirtualReality.js (Editor): *getToolbar ()* To edit the background image has been added to the toolbar an accordion called *background,* which contains two buttons.

```
basic: {
        __yam: "Background" , Icon: 'Crop_original' ,
        Buttons: {

            imagenBack: {
                __yam: '' , Type: 'Select' , Value: state.imagenBack, options: [ 'Choose a background ...' , '360_world.jpg' , 'Cloth-planets.jpg' , '



                        cloth-nature.jpg ' , 'Cloth-nature2.jpg' , 'Cloth-nature3.jpg' , 'Panoboom.jpg' , 'Cloth-people.jpg' ],

            },
            urlBack: {
                __yam: 'Search environment' , Type: 'External_provider'
                , Accept: "Image / *" , Value: state.urlBack,
                autoManaged: false ,


}}}
```

The first button, type *select,* offers the user a list of images prede fi ned with which already has the application. The second is a *external_provider,* Ediphy uses a component that implements the API repositories like Flickr or Vish for imaging (indicated they should be accepted only image files), as well as allowing load local files.

index.js The component responsible for selecting background, *Background.js,* It depends on the main component, *'Ediphy360'* the project is created to generate and staying in fi le *index.js.* He only method that implements is he *componentWillReceiveProps (nextProps).*

```
componentWillReceiveProps (nextProps) {
    if ( nextProps.imgBack! == Este . props.imgBack) {
        Environment.setBackgroundImage (asset (nextProps.imgBack) {format: nextProps.
                format}); } else if ( nextProps.urlBack! == Este . props.urlBack) {

        Environment.setBackgroundImage (nextProps.urlBack, {format: nextProps.format}); }

} Render () { return null }
```

Background.js not render anything directly, but uses the module *environement* 360 React to put a new fund each time you change the input data it receives. They have had to use two conditions because the syntax for a local file and to a past file by its URL, is different.

The audio environment

The other basic functionality of this application allows you to enable audio environment. To do this, from the toolbar you can choose whether to display the control buttons of the audio. Should you choose, the display buttons will be enabled and the user consuming the appeal may decide whether to play or stop. As already it explained how this check to be or not in the viewfinder, now you will see how it is built this functionality works.

VirtualReality.js (Editor): *getToolbar ()* In the accordion conguration it has added a button type *check box* display to indicate whether or not the control buttons of the audio.

```
__yam: "Configuration" , Icon: 'Build' , Buttons: {

        audioBack: {__name: 'Audio environment' , Type: 'check
        box' , Checked: state.audioBack,}
```

Client.js In *Client.js* React should indicate the components that will be rendered on the surfaces or in locations that have been created. For audio control, the buttons provided for this purpose are rendered on the surface available by default, which is associated with the main component *Ediphy 360 (* in *index.js).*

```
sup = const r360.getDefaultSurface (); sup.setShape
(Surface.SurfaceShape.Flat); sup.resize (200, 90); sup.setAngle (0, -0.4);

r360.renderToSurface (r360.createRoot ( 'Ediphy360' , { / * Initial props * / }),His p);
```

index.js To render these controls on the surface, a basic component of React 360 is used, the < *VrButton>,* which is used to wrap and provide a function that is released to the press the content, a component < *Text>* Indicating action.

```
render () {
          return (
              <View styles.panelForControls style = {}>
                  <Background imgBack = { Este . state.imgBack} = {urlBack Este . state.urlBack}
                          format = { Este . state.format} /> { Este . state.showAudio? (

                      <View styles.controls style = {}> <VrButton onclick = { Este ._ playaudio} {styles.button style =}>

                              <Text style = {styles.buttonText}> { 'Play' } </ Text> </ VrButton> <VrButton onClick = { Este ._ stopAudio}
                      {styles.button style =}>

                              <Text style = {styles.buttonText}> { 'Stop' } </ Text> </ VrButton> </ View>): null }


              </ View>);}
```

The function is called when the user selects the play button uses the native module *AudioModule* in case of being run from the viewer. All audio playback (environmental audio, sound effects unique and specialized audio) 360 React are controlled through this native module. For this application, the environmental audio is used and it is necessary to activate the method

playEnvironmental () exposing the module. Default is looped and the configuration must specify the values of the path to the audio source and volume. They also can change the Playback settings once this has started (default disable loop, mute the audio, etc). To stop the audio, simply it calls the method *stopEnvironemental ()* of the module.

The image carousel

Another feature that has this application is to have a central panel with the images that the user decides to load carousel mode. To do this, from the toolbar Ediphy you can select whether or not to enable this feature and upload the images you want to project (maximum 10). In the display, the user can use the side controls to scroll through the carousel.

VirtualReality.js (Editor): *getToolbar ()* In the accordion conguration it has added a button type *check box* to indicate whether to display or not the carousel. If activated, the other buttons are displayed: one to choose the number of images you want to load and another type *external_provider* for each uploaded image.

```
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 ,




    };} / * Setup toolbar other                                                        functionality * /
        showPanel: {
            __yam: 'Panel Aux' , Type: 'check box' , Checked:
          state.showPanel,}, numberOfPictures: {


            __yam: 'Number of pictures' , Type: 'Number' , Min 1,
            max: 10,


            hide: state.showPanel, value: state.numberOfPictures,}, ...
         urlPanels,
```

Client.js This component, called Spotlight, another surface is created during initialization at runtime.

```
import { Location, ReactInstance, Surface} from 'REACT-360-web' ; const photosPanel = new Surface (700, 600,
Surface.SurfaceShape.Cylinder);
// Code for all other functionalities
r360.renderToSurface (r360.createRoot ( 'Projector' ), PhotosPanel,);
```

A being a root component, you must register it with *AppRegistry.registerComponent ()* in the fi le *index.js* to associate with the surface.

ProyectorComponente.js This component is responsible for rendering the surface to which it is associated with the selected images in the toolbar user. As explained in the section of the *ConexionModule,* the state is updated with the data sent Ediphy. In the event that the control is activated,

showPanel It will be true and the projector will be rendered on the surface. In *arrayImgs* the directions of the images provided in the toolbar are saved, and
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();}});};
```

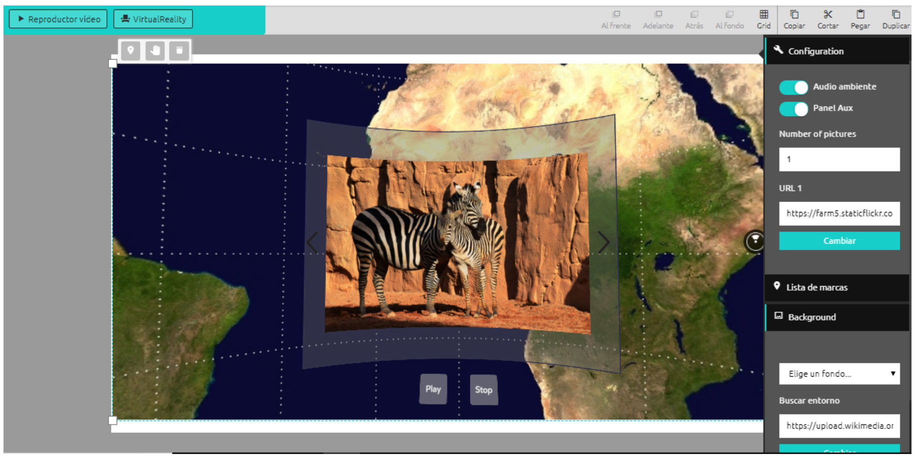
La función encargada de comunicarse con Ediphy para las marcas que enlazan con otro contenido llama a su equivalente en el componente que agrupa a todas las marcas y este utiliza el módulo nativo de conexión que ya se ha explicado. Si es una marca de tipo *pop-up,* se utiliza la función *enVisor()* del 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.



Audio ambiente
Panel Aux
Number of pictures: 1
URL 1: https://farm5.staticflickr.co
Elige un fondo...
Buscar entorno: https://upload.wikimedia.or

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 . a ñadir 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.

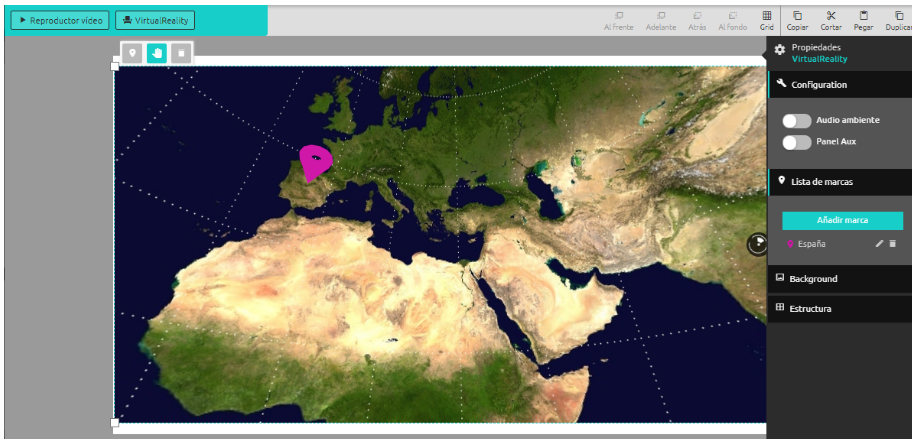


Figura 10 : Posicionar marcas por el entorno

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.

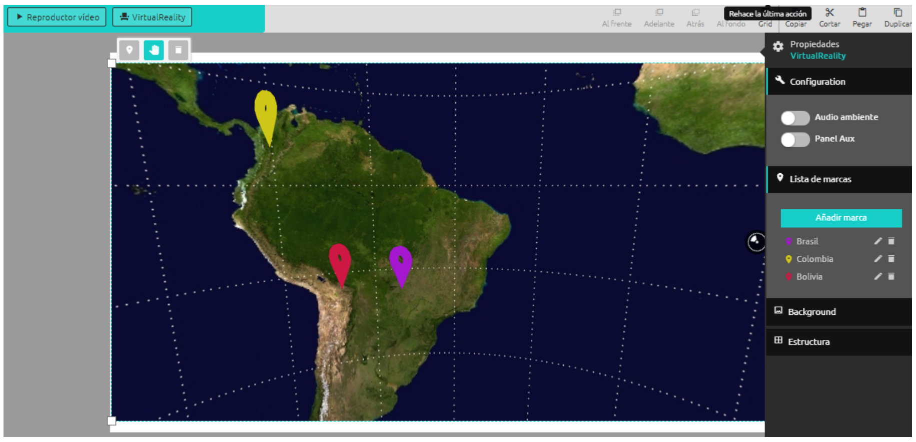


Figura 11 : Información en las marcas

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 *arn test"* o *arn 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 e. 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 e.

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 e 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 e/ 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 e/ hora.

Horas trabajadas Coste	de mano de obra total 300
	3750 e

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 e/ kWh. Por lo que quedaría un coste de 132 e.

F. Coste total

Concepto	Coste (e)
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.