UNIVERSIDAD
REY JUAN CARLOS

GRADO EN INGENIERÍA EN SISTEMAS DE TELECOMUNICACIÓN

Curso Académico 2018/2019

Trabajo Fin de Grado

GENERADOR DE GRÁFICOS PARA UN PORTAL LIFERAY UTILIZANDO REACT

Autor : Rubén Barrado Jiménez

Tutor : Gregorio Robles Martínez

Dedicado a todas las personas que me han ayudado a llegar hasta este momento, tanto familia, como compañeros y profesores.

II

Agradecimientos

Mi mayor agradecimiento es a mi familia, quienes han sido capaces de aguantar mis peores momentos durante esta carrera. Han sido capaces de darme ánimos en los peores momentos y de alegrarse, incluso más que yo, en mis mejores momentos y logros de esta dura carrera. Les agradezco todo tipo de esfuerzo, tanto moral, como económico, que han hecho para que haya llegado hasta aquí.

Por otro lado, agradecer a mis compañeros con los que he pasado muchos años de mi vida y una etapa muy importante, sin ellos esto hubiera sido más difícil todavía. Han hecho que las clases, las tardes de estudio, las prácticas y, en general, la carrera sea más divertida.

Agradecer a la empresa *Everis* por la oportunidad que me ha dado y las facilidades para poder desarrollar este proyecto y poder formarme como profesional en esta etapa. También a Gregorio por todas las facilidades a la hora de ayudarme con este trabajo.

Por último, agradecer a todos los profesores con los que me he cruzado por la carrera, me han ayudado a adquirir unos valores y unos conocimientos que no tenía.

Resumen

El presente trabajo de fin de grado ha consistido en integrar soluciones de uso extendido en los portales web que suelen ser utilizados por las compañías, empleando el framework de React dentro de la plataforma de Experiencia Digital de Liferay. Para ello se propone la integración de un gestor de contenidos, como es Liferay, con las bibliotecas y funciones de JavaScript, utilizando React.

Para llevar a cabo la integración se ha realizado el desarrollo de un portlet o componente, en el cual, a partir de unos datos de entrada, en tiempo real y en formato *JSON*, podamos generar tipos de contenido útil para la web y sus usuarios. En dicho componente podemos ver diferentes datos mostrados en forma de gráficos utilizando bibliotecas que permiten la creación de gráficos con React. Los datos utilizados en este trabajo de fin de grado corresponden al tiempo diario y semanal en las capitales de las comunidades autónomas de España.

Este proyecto se ha llevado a cabo en colaboración con la empresa *Everis*, para poder solucionar los inconvenientes causados al intentar implementar funciones JavaScript en sus proyectos realizados con Liferay.

Summary

This final project has consisted of integrating solutions of extended use in web portals that are usually used by companies using the React framework within Liferay's Digital Experience platform. For this purpose, it is proposed to integrate a content management system, such as Liferay, with the JavaScript libraries and functions, using React.

To carry out the integration, I have developed a portlet or component, in which, from some input data, in real time and in JSON format, we can generate types of content useful for the web and its users. In this component we can see the data displayed by graphs using libraries that allow the creation of graphics with React. The data used in this final project correspond to daily and weekly weather in the capitals of the Spanish regions.

This project has been carried out in collaboration with the company Everis, in order to solve the inconveniences caused when trying to implement JavaScript functions in their projects made with Liferay.

Índice general

Índice de figuras

Capítulo 1

Introducción

Este Trabajo Fin de Grado tiene como principal objetivo la integración de bibliotecas de JavaScript, en este caso, React, en un gestor de contenidos, llamado Liferay para poder implementar páginas web con las características de ambas tecnologías. En este proyecto se ha realizado la integración mediante la generación de gráficos utilizando las bibliotecas "Chart.js", "React-chartjs-2" y "React-chartjs2", de React.

1.1. Contexto

Actualmente son muchas las empresas que se han decantado por usar gestores de contenidos para realizar sus páginas web propias o las de sus clientes. Esto es debido a que los gestores de contenidos Open Source son capaces de facilitar el trabajo y poder llevar a cabo páginas web que pueden ser controladas por personas que no tienen conocimientos informáticos.

Los gestores de contenidos permiten la creación, manipulación y publicación de contenidos en una página web de manera muy sencilla y personalizable. Se pueden implementar nuevas funcionalidades una vez la web está operativa para los usuarios sin la necesidad de intervención de un desarrollador.

La mayoría de las compañías emplean estos gestores de contenidos para crear la Intranet de usuarios de sus clientes debido a que proporciona ayuda a la hora de desarrollar soluciones empresariales, con resultados inmediatos y a largo plazo, ofreciendo un framework de desarrollo capaz de adaptarse a las necesidades requeridas.

El principal inconveniente de emplear este método para la creación de una Intranet es que el código fuente es más complejo que emplear un desarrollo a medida para dicha página. Este inconveniente lo podemos evitar implantando JavaScript y sus bibliotecas en el gestor de contenidos.

En un gestor de contenidos cada página del portal actúa como un contenedor de diferentes aplicaciones, y si, por ejemplo, en una página web hay varias subpáginas se podría usar el mismo contenido en todas ellas si fuera necesario, además de poder eliminarlo de una subpágina en un momento dado o cambiarlo de lugar. De este modo, al tener integrado React con el gestor de contenidos, se puede añadir la web creada con React en cualquier parte de nuestro portal, incluso se podría tener la web en diferentes subpáginas sin necesidad de repetir el código.

Los portlets con React o con JavaScript se pueden desplegar en Liferay por otros medios, como desplegarlos en otras plataformas y accediendo a ellos vía *API*[1], pero este método es más rápido y más sencillo de realizar debido a que desplegamos el portlet directamente, pudiendo ir probando nuestros cambios en React según vamos diseñando el componente.

1.2. Motivación

Este proyecto ha sido realizado en colaboración con la empresa *Everis*[2], en la que estoy trabajando actualmente. He trabajado en dicha empresa en varios proyectos llevados a cabo por el gestor de contenidos de Liferay, muy utilizado en el mundo empresarial como base de portales públicos y privados, tales como intranets o portales de cliente. Por norma general, todos los componentes desarrollados sobre la tecnología Liferay se desarrollan en Java.

La integración de React en Liferay era un trabajo pendiente de hacer que podría incluir grandes mejoras en los proyectos con las que trabaja la empresa, por lo que ha sido una gran oportunidad de aprendizaje en dicha empresa. React es una de las más populares bibliotecas de JavaScript. En portales de Liferay, React sería muy importante a la hora de crear componentes debido a la velocidad y fluidez de las peticiones bajo demanda. Estos componentes son

[1] Application Programming Interface

[2] `https://www.everis.com/spain/es/home-1`

reutilizables en todo tipo de portales y proyectos desarrollados por Liferay, lo que es una gran ventaja.

React es una de las plataformas más utilizadas por las empresas, por encima de *Angular*, que hasta hace poco dominaba el mercado por encima de React, por lo que muchas empresas piden desarrollos llevados a cabo con React, por lo que era una buena oportunidad para incorporar esta biblioteca de JavaScript en la plataforma Liferay.

Como hemos visto, React es un framework que está ganando terreno y está incrementando su uso en el mercado, tal y como podemos ver en la Figura 1.1, en la cual podemos ver la comparativa, realizada por Jobfluent[3], entre las bibliotecas de React, Angular y VUE, donde podemos ver el crecimiento de React respecto al resto de frameworks.

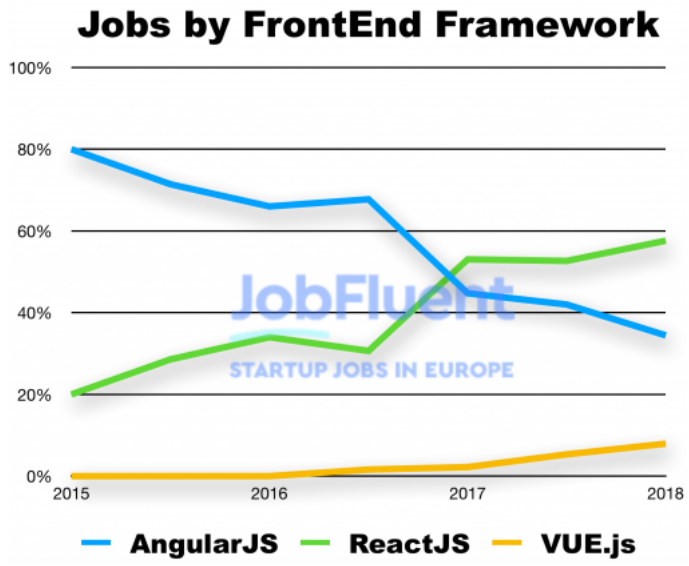

Figura 1.1: Comparativa entre las bibliotecas más populares de JavaScript en el mercado

Por tanto, la implementación de este proyecto plantea como principal reto la utilización de dos tecnologías extendidas en el mercado y que habitualmente no se utilizan de forma conjunta, por un lado cubre las necesidades de las empresas que apuestan por gestores de portales como Liferay, y otro responde a la implantación de React como tecnología asentada en el mercado y en constante evolución.

[3] `https://www.jobfluent.com/es`

1.3. Estructura de la memoria

En este apartado se va a describir la estructura de la memoria, que se ha dividido en 6 capítulos explicados a continuación:

1. *Introducción*

   En este primer capítulo se explica el contexto en el que se ha desarrollado, su motivación y la estructura de la memoria.

2. *Objetivos*

   En este apartado se detallan los objetivos del proyecto.

3. *Estado del arte*

   En este capítulo se explica el estado del arte de las tecnologías utilizadas en este trabajo.

4. *Instalación e implementación*

   Donde se explican los pasos necesarios para la instalación del entorno en el que se ha desarrollado el proyecto, así como su implementación.

5. *Resultados obtenidos*

   En este apartado se muestran los resultados obtenidos.

6. *Conclusiones*

   Por último, se detallan las conclusiones tras la consecución de los objetivos.

Capítulo 2

Objetivos

2.1. Objetivo general

El objetivo principal de este proyecto es integrar Liferay con el framework de React, pudiendo utilizar sus bibliotecas y funciones. Para ello ha sido necesario poder conocer Liferay y su funcionamiento respecto a módulos o portlets pudiendo configurar sus dependencias. Estas dependencias son esenciales a la hora de conectar el gestor de contenidos con React, debido a que al ser una biblioteca de JavaScript no es posible su utilización dentro de Liferay.

La utilización sólo es posible a partir de la versión 7.1 de Liferay, en la cual tenemos un plugin que permite la integración. Hay varias formas de realizar dicha integración:

- Configurando las dependencias a mano y modificando los ficheros necesarios para poder conectar React.

- Con una serie de instrucciones *npm*[1] en los cuales se configuran las dependencias automáticamente.

En este caso hemos optado por la segunda opción debido a que es más rápida de configurar y más asequible, aunque sea necesario realizar varios pasos para la instalación de todos los paquetes necesarios, dicha instalación se explicará en el capítulo 3.

[1] `https://www.npmjs.com`

2.2. Objetivos secundarios

En el camino hasta conseguir el objetivo principal, ha sido formarme en la biblioteca React y con el gestor Liferay. Este aprendizaje me ha resultado interesante debido a dos razones:

1. La formación en Liferay debido a la posibilidad que te ofrece empresarialmente tener conocimientos sobre este gestor, ya que es altamente utilizado por las empresas de este sector.

2. La formación en JavaScript, y en especial en la biblioteca React, debido a que últimamente está siendo muy demandado por las empresas y muy utilizado en el mercado.

Otro de los subobjetivos a la hora de realizar este proyecto ha sido poder familiarizarme con las dos tecnologías debido a que en muchos proyectos utilizar ambas tecnologías sería de gran ayuda.

Capítulo 3

Estado del arte

En este apartado se va a hablar de las tecnologías utilizadas para la realización de este proyecto. Dichas tecnologías son Liferay, React y JSON[1].

Para poder hablar de Liferay, primero debemos tener unos conceptos claros sobre el significado de un gestor de contenidos.

3.1. Gestor de contenidos

Un gestor de contenidos o *CMS* (Content Management System) es una aplicación a la que accedemos a través de un navegador que permite crear, manipular y organizar contenidos dentro de una página web sin ser necesario tener conocimientos informáticos. Estos contenidos son administrados por los usuarios con roles determinados dentro de la página web [8].

Los pasos que hay que seguir para tener un contenido web son:

1. *Creación del contenido web*: generado por editores integrados en los que se diseña el contenido como necesite el usuario.

2. *Publicación del contenido web*: una vez se ha creado el contenido web podremos publicarlo y tenerlo disponible para usarlo.

3. *Gestión del contenido web*: una vez publicado el contenido estará disponible para usarlo en la zona de la web que queramos, pudiéndolo cambiar de lugar, editarlo o eliminarlo.

[1] JavaScript Object Notation

Los contenidos puedes ser organizados en varias páginas.

4. *Presentación del contenido web*: podremos ver su resultado final desde cualquier navegador.

Un *CMS* tiene varias características [11]:

- Proceso de creación de contenido rápido, intuitivo, dinámico y fácil, donde no es necesario tener conocimientos informáticos.
- Gran flexibilidad, velocidad, rendimiento y seguridad de la web.
- Posee medios de soporte a los usuarios y administración del sitio sencilla.
- Reducción en los costes de mantenimiento.
- Gestión de las subpáginas que puede haber en una web y el multilenguaje.
- Soporte en varios dispositivos y canales.

Aparte, debe cumplir varias funciones para poder ofrecer contenidos de alto nivel que son:

- Herramientas que permitan crear contenidos como plantillas, estructuras y workflows.
- Interoperabilidad con otras tecnologías para poder obtener un mejor resultado.
- Capacidad de personalización en todos los contenidos.
- Integración con los sistemas de entrega.

Los gestores de contenidos pueden ser de diferentes sistemas, para el caso de Liferay es un sistema de Experiencia Digital *DXP*[2]. La principal función de un sistema *DXP* es la de proporcionar un contenido adaptable a cualquier canal digital en la creación del contenidos, además de mejorar las capacidades de negocio por medio de la digitalización. Últimamente, todos los *CMS* se están convirtiendo en *DXPs* debido a que las necesidades de las empresas son digitales y con *DXP* se pueden aprovechar las experiencias de los clientes de un *CMS* y añadirla a las capacidades de integración.

[2]Digital Experience Platform

3.2. Liferay

Este Trabajo Fin de Grado es un proyecto de Software Libre, lo que significa que es un programa que ofrece una serie de libertades a la hora de modificar el código, estudiarlo y utilizarlo en cualquier sitio y para cualquier propósito.

Al ser un proyecto desarrollado con Liferay, debe cumplir las libertades que ofrece un Software Libre. Esto implica que se permite el acceso y la modificación del código para el beneficio de su comunidad.

Liferay es un portal de gestión de contenidos Open Source basado en Java EE orientado a la solución y creación de portales web empresariales con resultados inmediatos y a largo plazo. Se basa en una arquitectura modular que proporciona herramientas de desarrollo. Se trata de reutilizar marcos y bibliotecas, a la vez que permite crear los propios [6].

Liferay puede utilizarse con la mayoría de servidores de aplicaciones, contenedores de servlets (en nuestro caso Tomcat) y base de datos (*MySQL*, *Oracle*).

En Liferay hay una comunidad con un foro, donde los usuarios exponen sus dudas para ser resueltas. Las características de Liferay son:

- Facilitar la creación, gestión y administración de portales web, con la colaboración de sus usuarios.
- Cada página del portal actúa como un contenedor de diferentes portlets, que puede ser una línea de texto o, como en el caso de este trabajo, una aplicación capaz de obtener datos en tiempo real de una *API* y mostrarlos en forma de gráficos.
- Estabilidad para garantizar la calidad de la aplicación.
- Seguridad.
- Eficiencia y rendimiento.
- Escalabilidad y durabilidad.
- Portabilidad, usabilidad y configurabilidad.

Puede utilizarse para [5]:

- Desarrollar aplicaciones completas (portlets o módulos).
- Personalizar una aplicación.
- Crear nuevos servicios web para usarlo con un sistema externo, como por ejemplo un móvil.
- Desarrollar aplicaciones móviles con Liferay Screens.
- Desarrollar temas de apariencia personalizados.

Todas estas utilidades se desarrollan como módulos en archivo *JAR*[3] que siguen estándares *OSGi*[4], que es un framework usado para desarrollar aplicaciones Java, que mantienen los plugins en sistemas *WAR*[5].

Un módulo o portlet, es un componente que puede usarse para crear aplicaciones de gran nivel mezclándose entre ellos y combinándose. Liferay es usado para desplegar los módulos y así crear un portal con varios contenidos web.

Un portal es una plataforma donde se construyen aplicaciones y sitios web, con Liferay podremos tener un portal con múltiples características en los que podemos desarrollar gran variedad de aplicaciones gestionando sus usuarios, creando varios sitios o páginas que necesitan poco mantenimiento, y creando entornos colaborativos.

Los portales se dividen en sitios compuestos por páginas donde se muestran los contenidos (portlets). Liferay puede asignar roles a determinados usuarios o grupos de usuarios que pueden modificar los contenidos, eliminarlos, crearlos y organizarlos. Estos usuarios tienen una serie de contenidos y funcionalidades que les aporta Liferay por defecto, que son:

- Gestión de contenidos (visor de contenido web, publicador de contenidos, galería multimedia, documentos y multimedia, aplicaciones anidadas).
- Menú de navegación.

[3] Java ARchive
[4] Open Services Gateway initiative
[5] Web Application Archive

- Búsqueda (barra de búsqueda, opciones de búsqueda, sugerencias, resultados).
- Colaboración (blogs, calendarios, formularios, foros).
- Comunidad (comentarios, sitios web, valoraciones, notificaciones).
- Herramientas (búsqueda web, login, selector de idioma).
- Noticias (alarmas, avisos, redes sociales).
- Social (actividades, contactos, miembros, perfil).
- Wiki (visor de la wiki, wiki, menú de árbol de la wiki).

Todos estos tipos de contenidos pueden ser usados por los usuarios sin necesidad de realizar ningún desarrollo informático, simplemente debe estar creado el contenido del tipo que queramos usar y moverlo a la zona del portal que deseemos.

Si fuera preciso, como en el caso de este proyecto, de crear algún tipo de contenido que no tenemos por defecto, podremos crear portlets y gestionarlos.

Liferay es un gestor de contenidos que permite instalar y gestionar portlets. Estos portlets pueden ser movidos a cualquier página creada en el portal, pudiendo introducir todos los portlets que queramos en cada página, como ocurre con los contenidos que tiene Liferay por defecto.

Es una plataforma que ofrece crear aplicaciones web, móviles y servicios rápidamente. Una de las características de Liferay es la versatilidad para crear todo tipo de sitios web como intranets, entornos colaborativos y aplicaciones móviles. Se puede implementar con contenedores de *JavaEE*[6] y otros servidores admitiendo una gran variedad de base de datos sin necesidad de cambiar el código fuente.

Este gestor de contenidos es Open Source, o código abierto, que sigue un modelo de desarrollo colaborativo y, además, se basa en los estándares, lo que hace reducir el bloqueo en Liferay.

Para la realización de este proyecto es necesario tener una versión de Liferay 7.1 o superior, debido a que es la versión que soporta el estándar JavaScript, gracias a *Babel*[7] y a un plugin

[6] Java Enterprise Edition

[7] `https://babeljs.io/`

que se puede instalar en el *MarketPlace*[8] de Liferay, llamado *Liferay JS Portlet Extender*[9]. Este plugin está disponible a partir de la versión 7.1, y se activa a través de *Marketplace* en el panel de control de Liferay, por lo que en las versiones anteriores no se podía usar el estándar JavaScript. Además de éste, Liferay soporta varios estándares importantes [2]:

- Portlets 1.0 y Portlets 2.0 (JRS-168 y JRS-286): permite la ejecución de cualquier portlet.
- JSF (JRS-127, JRS-314 y JRS-344): permite crear aplicaciones web basadas en componentes.
- EcmaScript 2015: permite el estándar JavaScript. Este estándar es el que nos permite realizar el proyecto.
- Servicio de interoperabilidad de gestión de contenido (CMIS): permite que los documentos de Liferay se comporten como una interfaz con cualquier repositorio externo.
- Repositorio de contenido Java (JSR-170).
- JAX-RS y JAX-WS: permite la creación de servicios web.

Liferay se basa en varias tecnologías importantes [5]:

- Para Front-end utiliza *Metal.js*, *Bootstrap* y *Sass*.
- Para Mobile utiliza Liferay Screen, Android e iOS.
- Para Back-end utiliza *OSGI*, *Spring* (para transacciones), *JavaEE*, *Hibernate* (acceso a la base de datos), *Ehcache* (almacenamiento en la caché) y *Elasticsearch* (para búsqueda).

El gestor de contenidos Liferay tiene una arquitectura que debe cumplir que el producto sea utilizable, rápido, robusto y personalizable. La arquitectura de Liferay es modular, como se ha dicho anteriormente, esto hace que los módulos cumplan con sus definiciones, implementaciones, prioridades y dependencias.

[8] `https://web.liferay.com/es/marketplace`

[9] urlhttps://web.liferay.com/es/marketplace/-/mp/application/115543020

Al tener una arquitectura modular y basada en componentes, al tener un problema, en vez de cambiar la aplicación entera para arreglarlo, en el caso de tener una arquitectura monolítica, simplemente deberíamos modificar el módulo que está teniendo problemas para solucionar el error, permitiendo además, poder volver al inicio en el caso de no ser necesaria ninguna modificación.

Una aplicación basada en componentes ayuda al desarrollador en:

- Poder escribir la aplicación en paralelo con otros desarrolladores que realizan un componente diferente.
- Poder extender la aplicación creando nuevos componentes.
- Poder habilitar y deshabilitar los componentes.

El tipo de web más utilizado con Liferay es la creación de Intranets, ya que ofrece una gran capacidad de comunicación personalizada para cada usuario, debido a [13]:

- Workflows personalizados.
- Experiencia de usuario moderna.
- Gestión de activos digitales.
- Contenidos modificables por usuarios dependiendo de su rol.
- Integración con sistemas existentes utilizando Liferay DXP.
- Capacidad de crear un entorno móvil.
- Unifica los sitios corporativos de los usuarios mediante un login.

3.2.1. MVC

Las aplicaciones web se suelen desarrollar siguiendo un patrón. El patrón que sigue Liferay es el Modelo Vista Controlador (MVC), cuyas características son [5]:

- Mejorar la escalabilidad.

- Facilitar el mantenimiento.
- Promover la reutilización.

El patrón MVC se divide en:

- Modelo: contiene los datos de la aplicación y la lógica para usarlos.
- Vista: se encarga de la presentación de los datos.
- Controlador: se encarga de controlar los datos entre la vista y el modelo. Gestiona lo lógica de la aplicación actuando de intermediario entre modelo y vista.

El MVC es usado por la gran mayoría de las aplicaciones web, pero Liferay ha desarrollado su propio MVC, que se diferencia en:

- Ser más ligero.
- No tiene la necesidad de mantener los archivos sincronizados.
- Evita escribir código repetido.
- Oculta la complejidad de los portlets y facilita las operaciones comunes.

3.2.2. Portlet

Una vez hemos introducido Liferay, es necesario conocer el concepto de portlet, que es lo que se ha desarrollado en este trabajo. Un portlet es un componente modular reutilizable de un portal web. Liferay es un contenedor de portlets que permite la creación, utilización y eliminación de cada uno de ellos, pudiendo ser utilizados en cualquier portal. Sus características son [1]:

- Generan contenido dinámicamente.
- Interactúan con el portal mediante peticiones y respuestas.
- Tiene un ciclo de vida controlado por el contenedor.
- Son gestionados por el contenedor.

- No pueden generar contenido arbitrario.
- El contenedor procesa sus solicitudes.

Al interactuar con el portal en las peticiones y las respuestas, el portlet en las respuestas devuelve contenido que es mostrado en los navegadores. La diferencia entre un portlet y otros tipos de navegadores web es que los portlets se ejecutan en una parte de la página web, esto hace que el resto de la página este controlado por otros componentes.

Se colocan en las páginas por los usuarios que tienen los roles correspondientes o los administradores. Como se ha dicho anteriormente, en una página puede haber muchos portlets diferentes, pudiendo cambiar su colocación y su diseño en cualquier momento. En una página se pueden configurar diferentes portlets y a cada uno le podemos poner el tamaño que queramos, dependiendo de la configuración que tenga la página en la que se coloquen.

Al igual que Liferay, los portlets están basados en estándares, y además de los estándares de Liferay de Portlets 1.0 y 2.0, recientemente se ha lanzado el Portlet 3.0 que permite la evolución del portlet y su portal.

Los portlets manejan solicitudes en diferentes fases. Sus fases son [7]:

- Render: Genera el contenido del portlet. Si se ejecuta esta fase en un portlet, también lo hace con el resto de portlets en la página. Esta fase se vuelve a generar cuando se completa la fase Action y Event.
- Action: Cuando se genera una acción, se actualiza el estado del portlet
- Event: Procesa los eventos generados en la fase Action.
- Resource-serving: Sirve un recurso independiente del ciclo de vida, lo que permite a un portlet ser dinámico sin necesidad de ejecutar las fases anteriores. En esta fase se manejan las solicitudes *AJAX*.

Los portlets tienen, además, diferentes modos que indican la función que se está realizando en él:

- Modo de visualización: es el modo que permite acceder a la funcionalidad principal del portlet.

- Modo de edición: es el modo de configuración del portlet, permite personalizar el portlet.
- Modo de ayuda: es el modo que muestra la información de ayuda del portlet.

3.3. JavaScript

Para poder hablar de React, framework sobre el que está desarrollado el proyecto, es necesario saber unos conocimientos básicos de JavaScript. Es un lenguaje de programación, que está presente en muchos navegadores web, fue introducido en el año 1995 como una forma de añadir programas a una página web en el navegador *Netscape Navigator*. Desde entonces es un lenguaje muy utilizado, usado en el 90 % de páginas web, que hace posible aplicaciones web modernas y dinámicas [3].

A pesar de su nombre, JavaScript no tiene nada que ver con el lenguaje de programación Java. El nombre se debe a una estrategia de marketing debido a la importancia del lenguaje Java en esos momentos. Es un lenguaje de programación que no necesita ser compilado para probar su funcionamiento, simplemente se puede probar el código en el navegador directamente. JavaScript sigue el estándar *ECMAScript* para que sea independiente de cualquier empresa.

Las características de JavaScript son:

- Es un lenguaje relativamente fácil de aprender.
- Desarrollado por Netscape.
- Funciona en diferentes dispositivos y navegadores.
- Es un lenguaje rápido y ligero.

Es soportado por todos los navegadores, a partir de la versión *ECMAScript 2015*, y actualmente nos encontramos en la versión *ECMAScript 2016*.

3.4. React

Este proyecto de fin de grado está basado principalmente en conectar las tecnologías de Liferay y de React, para conseguir tener un portlet programado con el lenguaje de React en

Liferay.

React es una de las bibliotecas de JavaScript que permite manejar de forma eficiente la interfaz de usuario de las aplicaciones web. Su desarrollador fue *Facebook* y fue creado para resolver las propias necesidades de la compañía con el mantenimiento del código de los anuncios. Actualmente sigue siendo utilizado en *Facebook* para darle soporte a la página, así como realizar actualizaciones de forma permanente [10].

Sus principales ventajas son:

- Su gran ecosistema: Es una de las bibliotecas con mayor trayectoria, se generaron muchas bibliotecas extendiendo sus características y potencialidades, permitiendo realizar todo tipo de programas que nos propongamos.

- Su estabilidad y alta compatibilidad: Es una biblioteca que hace evolucionar todas sus versiones y sus bibliotecas que hace que no existan quiebres de compatibilidad entre ellas generando una gran estabilidad.

- Su rendimiento: Es una biblioteca liviana que genera tiempos de carga inicial muy rápidos. Además, la actualización de pantalla es muy rápida, lo que permite renderizaciones parciales e inteligentes mediante su Virtual DOM y el proceso de reconciliación.

Debido a estos motivos React es muy sólida y fundamentada. Representa, a nivel de arquitectura en el patrón MVC, la V del modo Vista, lo que hace que se pueda combinar con otras bibliotecas de JavaScript [12].

React es una biblioteca de código abierto, declarativa, eficiente y flexible que permite construir interfaces de usuario, componentes interactivos y reutilizables, con el objetivo de facilitar el desarrollo de *SPA*[10], obteniendo un gran rendimiento. Es una biblioteca que está mantenida por desarrolladores de *Facebook* e *Instagram*, al igual que por desarrolladores independientes. Los componentes interactivos se crean de forma sencilla diseñando vistas simples para cada usuario en la aplicación que hacen que tu código sea más predecible y más fácil de depurar.

[10] Single Page Applications

El nacimiento de React comienza en 2011 cuando Pete Hunt y unos desarrolladores de *Facebook* crean un port de *XHP* en una aplicación *SPA*. Este tipo de aplicación requiere un gran número de peticiones al servidor que *XHP* no conseguía resolver, por lo que los desarrolladores solicitaron usar *XHP* en el navegador con JavaScript, dando lugar a ReactJS.

Otras bibliotecas parecidas a React son *Meteor* y *Angular*, que permiten que se actualice de forma automática la interfaz de usuario cuando los datos de la aplicación cambian. La diferencia entre estas bibliotecas y React es la forma en la que se programa con React, debido a que se utiliza una programación orientada a videojuegos, mientras que el resto se realiza una programación orientada al sistema *"data-binding"*. Esta diferencia hace que con React tengamos una renderización parcial inmediata, mientras que el resto tiene una renderización persistente.

Al tener una renderización parcial e inmediata, se renderiza todo con cada cambio que se haga en el código de forma automática, por lo que React es capaz de autogestionarse. Esto se realiza mediante su Virtual DOM, donde está contenido la estructura de datos, el sistema de eventos sintéticos y el gestor de memoria. Con cada cambio en un componente de React, los pasos que se realizan son los siguientes [12]:

1. Se genera un nuevo árbol Virtual DOM.
2. Se compara con el Virtual DOM previo.
3. Se deciden cuáles son los cambios mínimos a realizar.
4. Se mandan los cambios a la cola.
5. Se procesan los cambios en el navegador.

En el Virtual DOM tenemos el sistema de eventos sintéticos, que quiere decir que React tiene su propio sistema de eventos que crea un único manejador de evento nativo en el nivel superior de la estructura de cada componente, por lo que no es necesario utilizar otras bibliotecas como *jQuery*[11], lo que hace que tengamos menos código que desarrollar. El sistema está normalizado por lo que es operable en todos los navegadores, además, es capaz de diferenciar entre eventos "*desktop*" (click con el ratón) y eventos móviles (táctil de la pantalla).

[11] `https://jquery.com/`

React ayuda a los desarrolladores a crear aplicaciones web que utilizan un gran intercambio de datos, preocupándose solamente de la interfaz de usuario de la aplicación. El código programado en React se realiza a través de componentes, que se renderizan automáticamente, como se ha comentado anteriormente. Una interfaz de usuario es creada a partir de componentes, que encapsulan el funcionamiento y la presentación. Los componentes se basan en otros componentes para solucionar las necesidades más complejas. Al estar basado en componentes, se pueden pasar datos a través de la aplicación sin modificar el estado fuera del DOM.

Otra característica de React es que es isomórfico, que quiere decir que con un mismo código se puede renderizar tanto en cliente como en el servidor disminuyendo la carga de trabajo para realizar aplicaciones web para buscadores. Todo esto se consigue gracias a *Node.js* y se puede reutilizar en la parte de presentación y en la lógica de negocio [9].

Esta biblioteca está siendo actualmente utilizada por grandes marcas como *Airbnb, Sony, Netflix, Yahoo, BCC, Dropbox* y, las ya comentadas, *Facebook* e *Instagram* entre otras.

React al ser una biblioteca de código abierto permite que sea modificada por cualquier persona con los suficientes conocimientos, para incluir mejoras o cambios. Estas modificaciones se pueden realizar modificando el código que subió *Facebook* a un repositorio de *GitHub* para facilitar a los desarrolladores poder implementar estos cambios.

3.5. JSON

Para poder realizar este trabajo ha sido necesario poder transformar los datos que venían en formato JSON y transformarlos en datos legibles. JSON (JavaScript Object Notation) es un formato de intercambio de datos. Se trata de una estructura de datos siguiendo las pautas del lenguaje de programación JavaScript. El resultado es un formato de texto que es independiente del lenguaje, pero que utiliza convenciones que son conocidos por los desarrolladores [4].

JSON fue definido por Douglas Crockford a finales de 2002. En poco tiempo el uso del formato incrementó, debido a su sencillez y facilidad de uso, y surgieron implementaciones para todos los lenguajes de programación.

Sus características son:

- Es un formato de datos, solo contiene propiedades.
- Requiere comillas para las cadenas y nombres de las propiedades.
- Pueden incluirse todos los formatos que sean válidos.

JSON es una notación ligera para almacenar y transportar datos que se utiliza cuando un servidor envía los datos a una página web. Es un lenguaje autodescriptivo y fácil de entender. Cuando se intercambian datos entre el servidor y la página web, estos datos pueden ser texto, lo que hace que podamos convertir el objeto en JSON, y enviarlo al servidor, o convertir el JSON en objeto para que pueda ser tratado, como en este caso, sin parsear ni traducir de manera difícil.

Es, asimismo, una notación fácil de leer y escribir por las personas, y fácil de interpretar y analizar para las máquinas. Es un lenguaje más pequeño y más rápido de analizar que el formato *XML*[12]. Estas propiedades hacen que sea una notación adecuada para el intercambio de datos.

En JSON existen varios tipos de elementos, entre ellos las matrices (arrays) y los objetos (objects).

- Un ejemplo de matriz sería: [“ruben”, “pepe”, “juan”]. Separados entre comas dentro de un corchete.
- Un ejemplo de objeto sería: {“nombre”: “Ruben”, “apellido”: “Barrado”, “universidad”: “Universidad Rey Juan Carlos”}. Son elementos que van definidos por “nombre: valor” separados por comas dentro de una llave.
- Una lista ordenada de valores: Es una mezcla entre las dos anteriores, llamados listas, vectores o secuencias.

Un ejemplo de JSON, que es parecido a lo que se ha usado en este proyecto, puede ser el que se ve en la Figura 3.1.

[12]eXtensible Markup Language

```
{"coord":
{"lon":145.77,"lat":-16.92},
"weather":[{"id":803,"main":"Clouds","description":"broken clouds","icon":"04
n"}],
"base":"cmc stations",
"main":{"temp":293.25,"pressure":1019,"humidity":83,"temp_min":289.82,"temp_m
ax":295.37},
"wind":{"speed":5.1,"deg":150},
"clouds":{"all":75},
"rain":{"3h":3},
"dt":1435658272,
"sys":{"type":1,"id":8166,"message":0.0166,"country":"AU","sunrise":143561079
6,"sunset":1435650870},
"id":2172797,
"name":"Cairns",
"cod":200}
```


Figura 3.1: Ejemplo de *JSON*

Capítulo 4

Instalación e implementación

En este capítulo se detallan los pasos que hay que realizar para tener configurado un entorno en el que poder hacer este proyecto.

4.1. Instalación de componentes necesarios

Para poder realizar este trabajo es necesario tener por un lado todos los paquetes y configuraciones que necesita Liferay, y por otro lado todos los paquetes e instalaciones que necesita React para ser conectado con Liferay.

Para utilizar Liferay con React, necesitamos tener un servidor con una versión de Liferay 7.0 o superior, debido a que es la versión a partir de la cual podemos instalar el plugin *JS Portlet Extender*.

En este proyecto se ha descargado el *tomcat* de Liferay 7.1.1[1] . Para poder arrancar el servidor, necesitamos tener una serie de configuraciones e instalaciones que nos permiten utilizar Liferay.

Lo primero que debemos hacer es descargar es el *JDK* y el *JRE*[2] de Java e introducirlos en las variables de entorno del sistema del ordenador de la forma que se muestra en la Figura 4.1.

[1]Descarga tomcat: `https://www.liferay.com/es/downloads-community`

[2]Descarga *JDK* y *JRE*: `https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html`

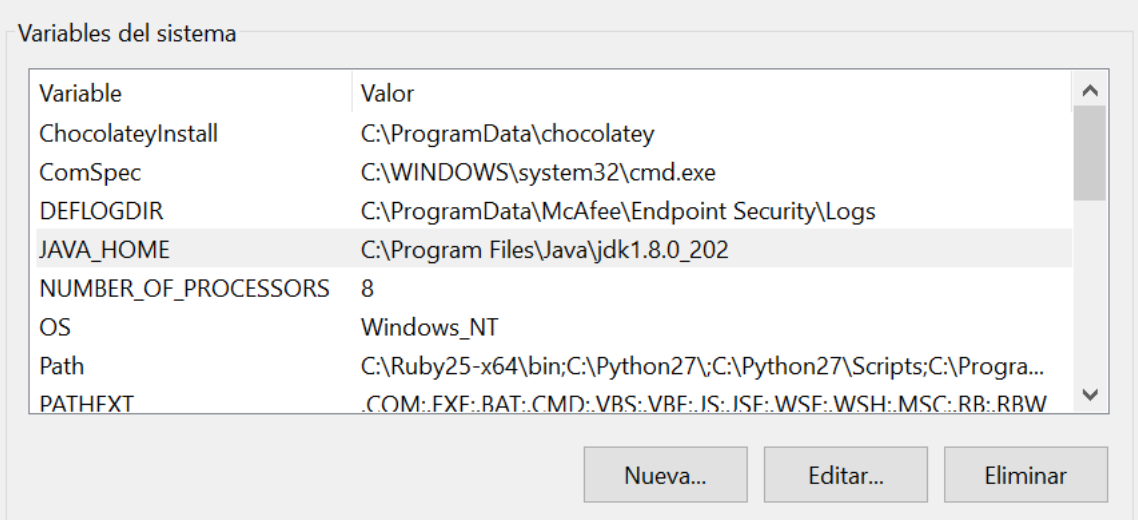

Figura 4.1: Ventana de configuración de las variables de entorno

Una vez configuradas las variables de entorno debemos instalar *Node.js*[3] , que es necesario para poder utilizar React. *Node.js* es un entorno de ejecución de código abierto para *JavaScript* construido por el motor de *Chrome* y orientado a eventos asíncronos. Proporciona un entorno de ejecución del lado del servidor que compila y ejecuta a velocidades muy altas. Se ejecuta para cada conexión y si no hay conexiones ejecutando *Node.js* no se ejecuta. Esto hace que no haya bloqueos de procesos debido a que no existen los bloqueos, lo que hace que se desarrollen sistemas escalables.

Con la instalación de *Node.js* obtenemos a la vez el sistema de gestión de paquetes *npm* que es necesario para instalar y manejar las dependencias del proyecto, así como las bibliotecas y los paquetes necesarios. Además de *npm*, necesitamos tener instalado otro manejador de paquetes llamado *Yarn*[4] , que se caracteriza por tener una velocidad mayor que *npm* debido a que permite descargas en simultaneo.

Para verificar que tenemos *Node.js*, *npm* y *Yarn* correctamente instalados, es necesario ejecutar las instrucciones en una consola como vemos en la Figura 4.2.

[3]Descarga *Node.js*: `https://nodejs.org/en/`
[4]Descarga *Yarn*: `https://yarnpkg.com/lang/en/`

```
C:\Dev\TFGProject>npm -v
6.9.0

C:\Dev\TFGProject>node -v
v10.13.0

C:\Dev\TFGProject>yarn -v
1.15.2
```

Figura 4.2: Instrucciones para verificar la instalación de los paquetes

Con estos pasos, tenemos todas las configuraciones y todos los paquetes necesarios instalados. Ahora es necesario arrancar el servidor de Liferay para tenerlo disponible, si es la primera vez que se ha arrancado, nos aparecerá una ventana de configuración para poder crearnos el usuario y su contraseña, además de poder configurar el idioma del portal. Una vez configurado el usuario y el idioma tendremos un portal disponible con la página principal como se ve en la Figura 4.3. Con esto tendremos a disposición todos los componentes y elementos que vienen configurados por defecto con Liferay. Podremos crear páginas y componentes para organizar el portal de la manera deseada.

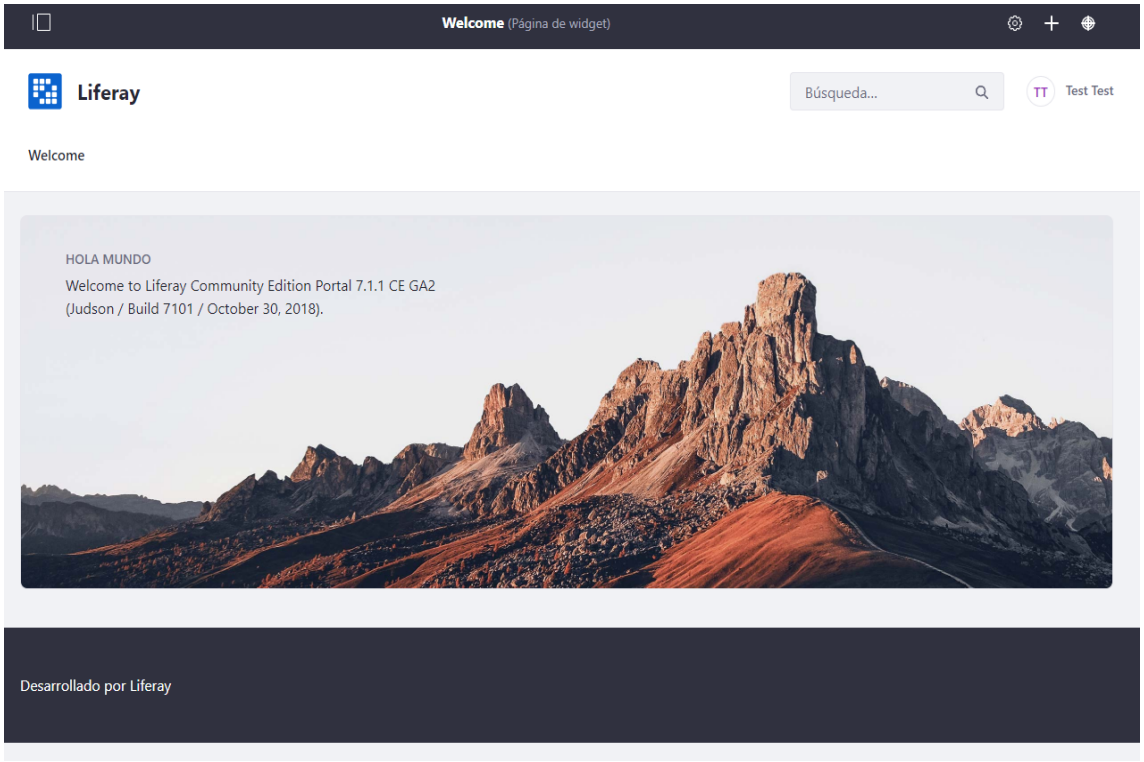

Figura 4.3: Visualización inicial de la página principal del portal de Liferay

4.2. Creación portlet React

Una vez tenemos un servidor Liferay arrancado y en funcionamiento, podemos empezar a integrar React con Liferay. Para ello es necesario instalar un generador de *Yeoman*[5], la instalación se realiza con la siguiente instrucción:

```
npm install -g yo
```

Con el generador de *Yeoman* instalado, el siguiente paso es instalar el generador de los paquetes de Liferay (liferay-bundle) de la siguiente manera:

```
npm install generator-liferay-bundle
```

Con ambos generadores instalados, ya podemos crear el portlet de React con la instrucción

```
yo liferay-bundle
```

que nos permite elegir el tipo de proyecto que queremos crear como se muestra en la Figura 4.4.

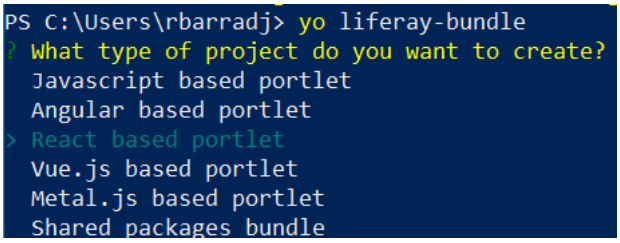

Figura 4.4: Elección del tipo de proyecto

El tipo de proyecto que queremos crear en este caso es "*React based portlet*". Una vez elegido, debemos configurar las opciones que se muestran en la Figura 4.5.

En la Figura 4.5 podemos ver que al crear el proyecto podemos darle el nombre que queramos, en este caso hemos elegido "portletreact", la descripción del proyecto y la categoría en la que queremos que aparezca en el portal de Liferay, en este caso "*Ejemplo*". Además, nos pregunta si tenemos alguna instalación local de Liferay, que es la que hemos realizado anteriormente, al decirle "*Yes*" nos pregunta sobre la localización de dicha instalación en la que habemos de poner la ruta en la que está el *tomcat* instalado en el apartado 4.1.

[5] `https://yeoman.io/`

```
PS C:\Users\rbarradj> yo liferay-bundle
? What type of project do you want to create? React based portlet
? What name shall I give to the folder hosting your project? portletreact
? What is the human readable description of your project? Portletreact
? Do you want to add localization support? Yes
? Do you want to add settings support?
  (□□ needs JS Portlet Extender 1.1.0) No
? Under which category should your portlet be listed? category.sample
? Do you have a local installation of Liferay for development? Yes
? Where is your local installation of Liferay placed? C:\Dev\TFGProject\liferay-ce-portal-7.1.1-ga2
? Do you want to generate sample code? Yes
   create package.json
   create README.md
   create .gitignore
   create .npmbuildrc
   create .npmbundlerrc
   create assets\.placeholder
   create features\localization\Language.properties
   create assets\css\styles.css
   create .babelrc
   create src\index.js
   create src\AppComponent.js
```

Figura 4.5: Creación portlet React

Al generar un código de ejemplo se generan los paquetes que vemos en la Figura 4.5 que nos permiten la conectividad entre React y Liferay. Por medio del fichero "*package.json*" se declaran las dependencias del portlet como se puede ver en la Figura 4.6. En el fichero "*index.js*" se reciben los campos "*contextPath*", que es la ruta del contexto donde se bajan los recursos estáticos del portlet, y "*portletElementId*", que es el identificador del nodo que permite la conectividad con el framework para generar la interfaz de usuario.

```
"devDependencies": {
    "babel-cli": "^6.26.0",
    "babel-loader": "^7.0.0",
    "babel-preset-env": "^1.7.0",
    "babel-preset-react": "^6.0.0",
    "copy-webpack-plugin": "^4.5.4",
    "liferay-npm-build-support": "^2.6.2",
    "liferay-npm-bundler": "^2.6.2",
    "webpack": "^4.0.0",
    "webpack-cli": "^3.0.0",
    "webpack-dev-server": "^3.0.0"
},
"dependencies": {
    "react": "^16.0.0",
    "react-dom": "^16.0.0"
},
```

Figura 4.6: Visualización del fichero "*package.json*"

4.3. Desarrollo de la aplicación

Una vez hemos creado el portlet y tenemos todos los paquetes necesarios instalados, es hora de comenzar a desarrollar la aplicación. En este apartado se van a explicar las configuraciones iniciales necesarias para poder usar las bibliotecas que permiten la generación de los gráficos, y las principales funciones utilizadas para obtener los datos que se van a mostrar.

El primer paso para desarrollar la aplicación es instalar las bibliotecas que nos permiten desarrollar la aplicación utilizando React. Para instalar las bibliotecas, recomiendo utilizar la herramienta *Yarn*, comentada en el apartado 4.1, debido a que no da conflictos con las dependencias ya instaladas, por el contrario, si usamos *npm* podemos llegar a tener algún tipo de conflicto.

Las primeras bibliotecas que debemos instalar son las que nos permiten programar utilizando *ECMAScript*. Se instalan con las siguientes instrucciones:

```
yarn add eslint
yarn add babel-preset-es2015
yarn add babel-plugin-transform-class-properties
```

Para verificar que se han instalado bien las bibliotecas, comprobamos que se han añadido al fichero *"package.json"* tal y como se muestra en la Figura 4.7.

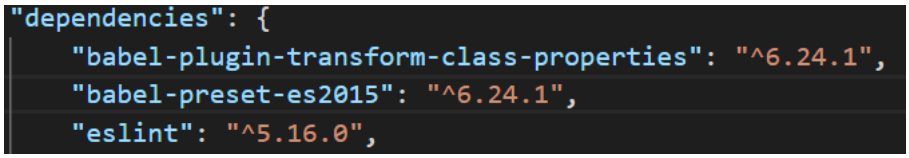

Figura 4.7: Visualización de las bibliotecas de *ECMAScript* en el fichero *"package.json"*

Una vez las bibliotecas están instaladas, debemos modificar el fichero "*.babelrc*" de la siguiente manera:

```
{
    "presets": ["es2015", "react"],
    "plugins": ["transform-class-properties"]
}
```

Con estas configuraciones ya se puede empezar a programar con React. El siguiente paso es instalar las bibliotecas necesarias para generar los gráficos y transformar los datos recogidos de la *API*. Estas bibliotecas son "*react-chartjs-2*", "*react-chartjs2*" y "*chart.js*". Se instalan usando las siguientes instrucciones:

```
yarn add react-chartjs-2
yarn add react-chartjs2
yarn add chart.js
```

Para verificar que se han instalado bien las bibliotecas, comprobamos que se han añadido al fichero *"package.json"* tal y como se muestra en la Figura 4.8.

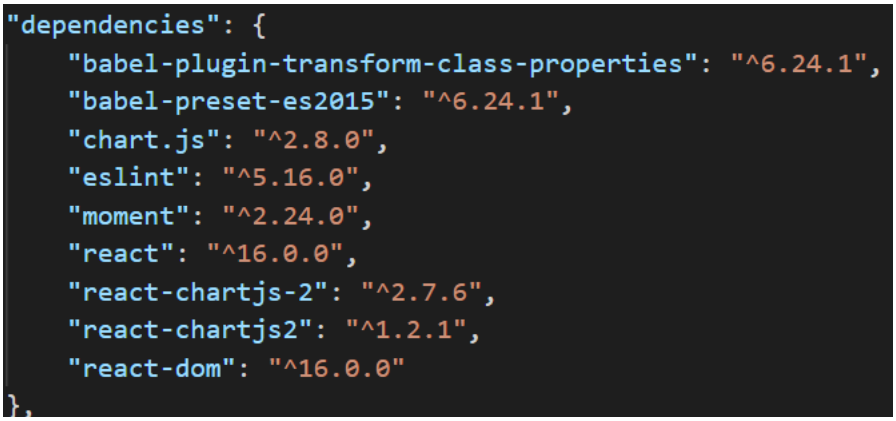

Figura 4.8: Visualización de las bibliotecas en el fichero *"package.json"*

Como podemos ver, con estas bibliotecas ya se puede empezar a generar gráficos. El siguiente paso es obtener los datos de la *API*.

En este proyecto se ha utilizado la *API OpenWeatherMap*[6], en la cual podemos obtener los datos meteorológicos para el día actual, y los datos meteorológicos de una semana. Estos datos vienen en formato *JSON*.

Para transformar estos datos en legibles, ha sido necesario emplear la biblioteca "*moment*", instalada con la siguiente instrucción:

```
yarn add moment
```

[6] https://openweathermap.org/

Para la generación de los gráficos se va a explicar como modo de ejemplo la obtención de los datos semanales. Los datos los obtenemos de la *API* de la forma que vemos en la Figura 4.9.

```
componentWillMount() {
    const api_key = "cf6b1fc1c9b7e2f6f5aa59ff2c25c965";
    const url = "http://api.openweathermap.org/data/2.5/forecast";
    const url_forecast = `${url}?q=${this.props.city}&appid=${api_key}`;

    fetch(url_forecast).then( resolve => {
        return resolve.json();
    }).then(forecastData => {
        let newWeather = this.transformForecast(forecastData);
        let weatherDay = this.obtainDataDays(newWeather);
        let dataBar = {
            labels: this.obtainDays(weatherDay),
            datasets: this.obtainDatasets(weatherDay)
        };
        this.setState({
            forecastData: forecastData,
            dataBar: dataBar
        });
    });
}
```

Figura 4.9: Función para obtener los datos de la *API*

En esta imagen podemos ver que obtenemos los datos mediante una url que accede a la *API* y recoge los datos en un JSON. Estos datos son transformados, en primer lugar, obteniendo los datos que queremos con la función que se muestra en la Figura 4.10.

```
transformForecast = (data) => (
    data.list.filter(item => (
        moment.unix(item.dt).utc().hour() === 3 ||
        moment.unix(item.dt).utc().hour() === 6 ||
        moment.unix(item.dt).utc().hour() === 9 ||
        moment.unix(item.dt).utc().hour() === 12 ||
        moment.unix(item.dt).utc().hour() === 15 ||
        moment.unix(item.dt).utc().hour() === 18 ||
        moment.unix(item.dt).utc().hour() === 21 ||
        moment.unix(item.dt).utc().hour() === 0
    )).map(item => (
        {
            weekDay: moment.unix(item.dt).format('dddd'),
            hour:  moment.unix(item.dt).utc().hour(),
            data: this.transformWeatherForecast(item),
        }
    ))
);
```

Figura 4.10: Función para transformar las horas y obtener datos del *JSON*

Esta función filtra los datos por las horas que necesitamos para obtener un gran número de

datos y así poder precisar la temperatura máxima y mínima. Para obtener los datos meteorológicos se usa la función *transformWeatherForecast* como se muestra en la Figura 4.11.

```
transformWeatherForecast (weather_data) {
    const {temp_min, temp_max, humidity} = weather_data.main;
    const temperature_min = this.getTemp(temp_min);
    const temperature_max = this.getTemp(temp_max);
    const {speed} = weather_data.wind;
    const wind = this.getWind(speed);
    const weatherState = this.getWeatherState(weather_data.weather[0]);

    const forecastData = {
            temperature_min,
            temperature_max,
            humidity,
            weatherState,
            wind
    }
    return forecastData;
}
```

Figura 4.11: Función para obtener datos meteorológicos del *JSON*

Como veíamos en la Figura 4.9, una vez transformados los datos de la *API*, necesitamos calcular la temperatura máxima y la temperatura mínima de cada día de la semana, para calcular estas temperaturas se usa la función *obtainDataDays*, donde se calculan las temperaturas de la forma que se ve en la Figura 4.12.

```
if(item.weekDay.includes(item_day.weekDay)){
    if(parseFloat(item.data.temperature_max) > parseFloat(item_day.data.temperature_max)){
        item_day.data.temperature_max = item.data.temperature_max;
    }
    if(parseFloat(item.data.temperature_min) < parseFloat(item_day.data.temperature_min)){
        item_day.data.temperature_min = item.data.temperature_min;
    }
```

Figura 4.12: Función para calcular la temperatura máxima y mínima de cada día

Esta función recorre los datos iterando según el día en el que estemos, obteniendo las máximas y mínimas temperaturas de los días de la semana, que se obtienen con la función *obtainDays* de la forma que se ve en la Figura 4.13.

Una vez hemos obtenido los datos de las temperaturas y los días, necesitamos introducir estos datos de tal manera que se muestren en los gráficos. Estos datos deben ir de la manera que se guardan en la función *obtainDatasets*, donde introducimos la temperatura máxima y mínima

```
obtainDays(data) {
    let days = [];
    data.map((item) => {
        days.push(item.weekDay);
        return days;
    });
    return days;
};
```

Figura 4.13: Función para obtener los días de la semana

en un objeto junto con su respectivo "*label*" que muestra la información de cada barra en el gráfico. La formación de este objeto lo podemos ver en la Figura 4.14.

```
obtainDatasets(data) {
    let object = [];
    let max = [];
    let min = [];
    data.map((item) => {
        max.push(item.data.temperature_max);
        min.push(item.data.temperature_min);
        return item;
    });
    object.push({
        label: 'Temperatura máxima',
        backgroundColor: '#D52B1A',
        borderWidth: 3,
        pointRadius: 4,
        data: max
    });
    object.push({
        label: 'Temperatura mínima',
        backgroundColor: '#6C6F70',
        borderWidth: 3,
        pointRadius: 4,
        data: min
    });
    return object
};
```

Figura 4.14: Función para obtener los datos que se muestran en los gráficos

Con esto podemos crear el objeto *dataBar*, que veíamos en la Figura 4.9. Este objeto se introduce en los gráficos de la forma que se muestra en la Figura 4.14, donde se crean los gráficos que veremos en el Capítulo 5.

```
<Bar
    id={'chart-Point'}
    width="800"
    height="350"
    data={this.state.dataBar}
    options={this.getOptionsBarMax('ºC')}
/>
```

Figura 4.15: Generación de los gráficos

4.4. Despliegue en Liferay

Tras realizar el desarrollo del portlet, es hora de desplegarlo y construirlo en el portal de Liferay, para ello debemos tener el servidor arrancado. Antes de desplegarlo hay que construirlo, ejecutando la siguiente instrucción, en la ruta en la que tenemos guardado el proyecto:

```
npm run build
```

Esta instrucción construye las rutas que hay en el proyecto y construye las dependencias que haya en el "*package.json*". La ejecución de la instrucción se puede ver en la Figura 4.16.

```
PS C:\Dev\TFGProject\portletreact> npm run build

> portletreact@1.0.0 build C:\Dev\TFGProject\portletreact
> babel --source-maps -d build src && npm run copy-assets && liferay-npm-bundler

src\AppComponent.js -> build\AppComponent.js
src\ChartDay.js -> build\ChartDay.js
src\ChartWeek.js -> build\ChartWeek.js
src\index.js -> build\index.js
src\LocationList.js -> build\LocationList.js

> portletreact@1.0.0 copy-assets C:\Dev\TFGProject\portletreact
> lnbs-copy-assets

Project assets copied.
Bundling 194 dependencies...
Bundling took 12s
Report written to liferay-npm-bundler-report.html
```

Figura 4.16: Ejecución de la instrucción *"build"*

Tras construir las dependencias y las rutas del proyecto, hay que desplegar el proyecto en Liferay con la siguiente instrucción:

```
npm run deploy
```

Esta instrucción realiza el despliegue del portlet a Liferay; su ejecución se muestra en la Figura 4.17.

```
PS C:\Users\rbarradj> cd C:\Dev\TFGProject\portletreact
PS C:\Dev\TFGProject\portletreact> npm run deploy

> portletreact@1.0.0 deploy C:\Dev\TFGProject\portletreact
> npm run build && lnbs-deploy


> portletreact@1.0.0 build C:\Dev\TFGProject\portletreact
> babel --source-maps -d build src && npm run copy-assets && liferay-npm-bundler

src\AppComponent.js -> build\AppComponent.js
src\ChartDay.js -> build\ChartDay.js
src\ChartWeek.js -> build\ChartWeek.js
src\index.js -> build\index.js
src\LocationList.js -> build\LocationList.js

> portletreact@1.0.0 copy-assets C:\Dev\TFGProject\portletreact
> lnbs-copy-assets

Project assets copied.
Bundling 194 dependencies...
Bundling took 13s
Report written to liferay-npm-bundler-report.html
Deployed portletreact-1.0.0.jar to C:\Dev\TFGProject\liferay-ce-portal-7.1.1-ga2
PS C:\Dev\TFGProject\portletreact>
```

Figura 4.17: Ejecución de la instrucción *"deploy"*

Como podemos ver, al hacer "*deploy*" se hace lo mismo que en "*build*", añadiendo el despliegue al *tomcat* configurado en el apartado 4.2 donde introducimos la ruta en la que lo teníamos.

Con el servidor arrancado, una vez se completa el despliegue, en la consola deberá aparecer el mensaje que se ve en la Figura 4.18.

```
2019-06-04 11:44:41.970 INFO  [fileinstall-C:/Dev/TFGProject/liferay-ce-portal-7.1.1-ga2/osgi/modules][BundleStartStopLo
gger:39] STARTED portletreact_1.0.0 [958]
```

Figura 4.18: Visualización de la consola del servidor al desplegar

Una vez vemos el mensaje "STARTED" dentro de la consola, el portlet estará disponible en el portal de Liferay, en la categoría que introdujimos en el apartado 4.2. En este caso, la categoría es "*Ejemplo*", por lo que, una vez logueados, seleccionamos la categoría en la sección de configuración de los contenidos de Liferay y arrastramos el portlet, de la forma que vemos en la Figura 4.19.

Posicionamos el componente en la zona que queramos del portal y tendríamos el portlet listo para visualizarse de la forma que veremos en el Capítulo 5.

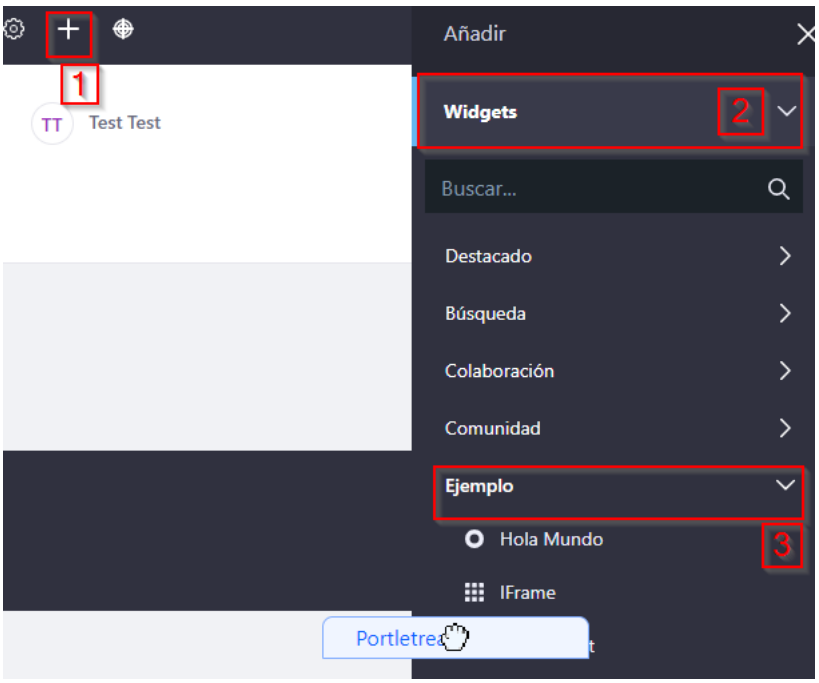

Figura 4.19: Despliegue del portlet en Liferay

Capítulo 5

Resultados obtenidos

Una vez se ha desplegado el portlet en Liferay y se ha movido a la zona donde se desea, podemos empezar a usar el componente. En este capítulo se van a detallar los resultados obtenidos al desarrollar e integrar la aplicación.

La arquitectura de la aplicación se muestra en la Figura 5.1, donde podemos ver que React se integra con Liferay mediante la herramienta *npm* y una vez en el portal de Liferay, realizamos llamadas a la *API* que nos devuelve los datos para poder generar los gráficos. Cada vez que se realiza un cambio en la aplicación, para mostrar otra ciudad o cambiar la visualización entre datos diarios y semanales, Liferay realiza la llamada a la *API* generando los nuevos gráficos.

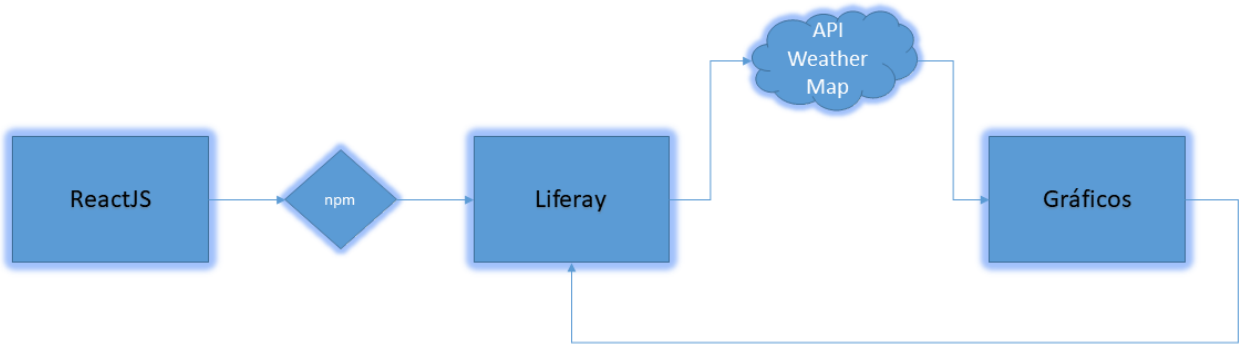

Figura 5.1: Arquitectura de la aplicación

5.1. Visualización inicial

La visualización inicial del portlet es la que se ve en la Figura 5.2, donde vemos una breve descripción del funcionamiento del componente, un selector para seleccionar una ciudad y un mensaje indicando que debemos elegir una ciudad.

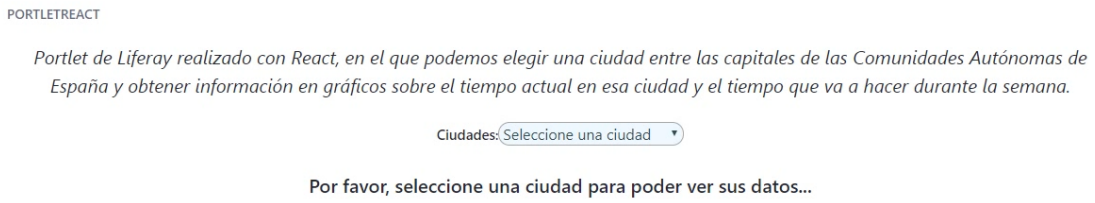

Figura 5.2: Visualización inicial del portlet

Si desplegamos el selector veremos una lista de ciudades, como se muestra en la Figura 5.3, en la cual debemos seleccionar una ciudad para mostrar sus datos, tal y como se verá en los siguientes apartados de este capítulo.

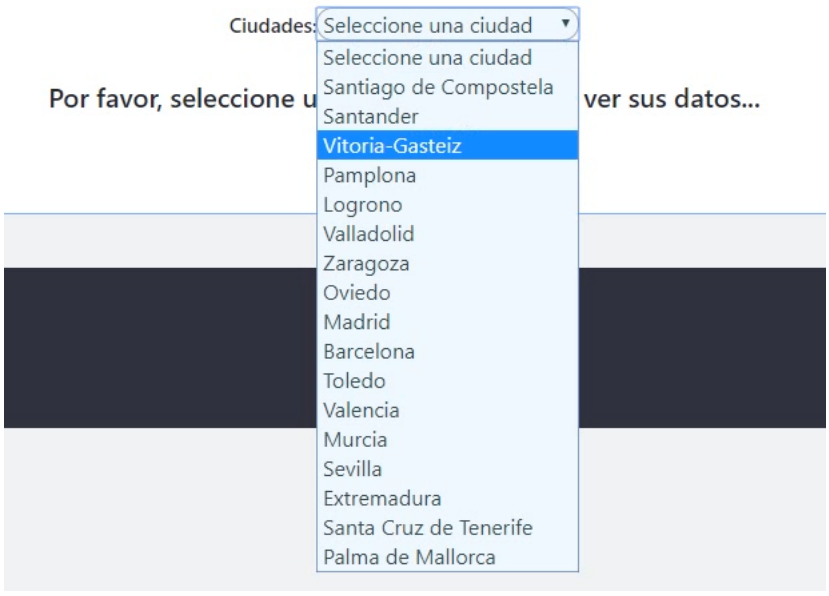

Figura 5.3: Visualización del selector de las ciudades

5.2. Visualización datos diarios

Al seleccionar la ciudad, se cargarán los gráficos con los datos diarios por defecto, como se ve en la Figura 5.4. En dicha imagen se pueden ver los datos de la temperatura, la humedad y el viento a unas determinadas horas del día. Estos datos nos los proporciona la *API* de tres horas en tres horas, de ahí que se muestren en los gráficos los datos cada tres horas.

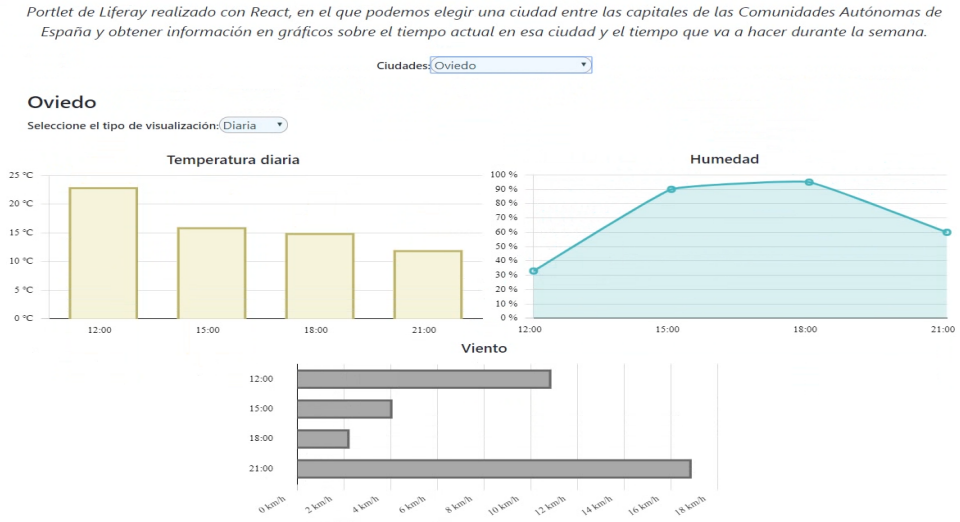

Figura 5.4: Visualización de los gráficos para los datos diarios

Para la temperatura diaria se muestran los datos cada tres horas en grados centígrados (^oC), para la humedad se muestra el porcentaje de humedad en esas horas, y para el viento se muestran los datos en kilómetros por hora (*km/h*).

Si nos posicionamos sobre alguna barra de algún gráfico, se verá información detallada sobre la hora en la que nos hemos posicionado, tal y como se muestra en la Figura 5.5.

En este caso, se ve información de la temperatura exacta a las 18:00. Si nos posicionáramos sobre otro tipo de gráfico, veríamos información detallada de la humedad exacta o del viento exacto en ese momento. Si se quiere cambiar la ciudad, se podría seleccionar otro ciudad y se cargarían los nuevos datos para la ciudad seleccionada.

En esta misma pantalla, hay un selector que permite seleccionar el tipo de visualización, como se ve en la Figura 5.6, eligiendo entre visualización diaria, como se ve en este apartado, o eligiendo visualización semanal, como se verá en el apartado 5.3. Si seleccionamos en el

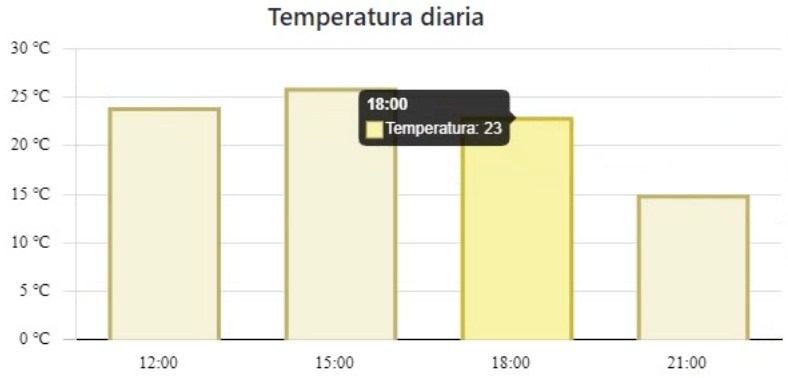

Figura 5.5: Visualización en detalle del gráfico

desplegable el tipo de visualización diferente a la visualización de la pantalla en la que nos encontramos, la página se carga y se muestran los datos del tipo seleccionado.

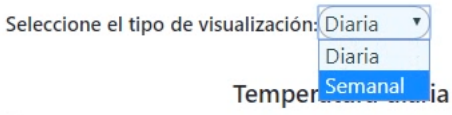

Figura 5.6: Selector para el elegir el tipo de visualización

5.3. Visualización temperatura semanal

Al seleccionar en el desplegable “Semanal” se cargan los datos para mostrarse el gráfico de la tempetatura semanal en una ciudad, tal y como se ve en la Figura 5.7.

En esta imagen se ven las temperaturas máximas, en rojo, y mínimas, en gris, que hay en la semana. De igual manera que pasa con los gráficos diarios, si pasamos por encima de alguna barra del gráfico se verá información detallada sobre dicha barra, indicando el día y la temperatura.

Desde esta visualización se puede ir a la visualización diaria, seleccionando en el desplegable del tipo de visualización la opción “Diaria”. También se puede cambiar la ciudad para poder comparar las temperaturas de distintas ciudades.

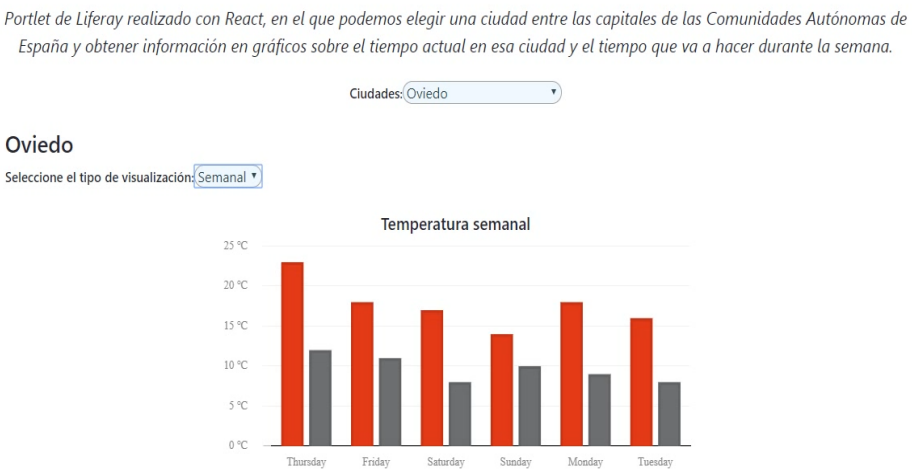

Figura 5.7: Visualización del gráfico para los datos semanales

Capítulo 6

Conclusiones

En este último capítulo se va a realizar un análisis de los resultados obtenidos, en base a los objetivos planteados, de la aplicación de los conocimientos aprendidos y de los posibles trabajos futuros relacionados con este proyecto.

6.1. Consecución de los objetivos

Si volvemos a los objetivos marcados para este proyecto, el objetivo principal era integrar React con Liferay, pudiendo utilizar sus bibliotecas y tener una aplicación de React dentro de Liferay para poder gestionarla de la manera que queramos. Este objetivo se ha cumplido como hemos podido ver en el desarrollo de este trabajo, pudiendo añadir y eliminar dependencias para crear la conexión entre Liferay y React, y pudiendo usar las bibliotecas de React dentro de la plataforma de Liferay.

Los objetivos secundarios eran objetivos de formación y familiarización con las tecnologías utilizadas. Estos objetivos, por mi parte, también se han cumplido, debido a que eran tecnologías que no había usado nunca y he podido defenderme con ellas y al final he podido conectarlas. He aprendido el manejo de las dependencias de Liferay y la navegación entre la plataforma, permitiendo diseñar portales a mi gusto, así como el manejo con las funciones y métodos de React y JavaScript, algo que considero muy importante viendo el mercado laboral que hay actualmente.

6.2. Aplicación de lo aprendido

Los conocimientos aprendidos me han servido para ser más independiente en la empresa en la que estoy trabajando, debido a que he aprendido a entender el comportamiento de Liferay con los portlets, algo que es muy importante a la hora de crear páginas web utilizando este gestor de contenidos.

Además me ha servido para poder enseñar a mis compañeros a añadir portlets realizados con bibliotecas de React, algo que no se había hecho en la empresa de la manera que se ha hecho en este proyecto.

Incluir bibliotecas de JavaScript en Liferay de esta manera, ejecutando instrucciones y desarrollando en JavaScript el código de la aplicación, hace que el desarrollo sea más rápido y se puedan realizar más pruebas sobre la aplicación, debido a que podemos ir probando la aplicación de JavaScript sin necesidad de desplegar en Liferay.

Esto es algo importante, ya que una vez estás realizando algún tipo de componente, para probarlo en Liferay hay que desplegar cada cambio que se haga y así poder visualizarlo, mientras que si empleamos JavaScript, podemos ver los cambios simplemente con guardar el código.

6.3. Trabajos futuros

En este proyecto se ha integrado React con Liferay, haciendo que el código programado en React se comunique y se visualice en Liferay.

Un posible trabajo futuro sería poder comunicar Liferay con React, no React con Liferay como se ha hecho en este trabajo. Esto es, que se pueda realizar un portlet con React, que al estar en Liferay se pueda configurar desde la configuración de Liferay, para comunicarle dichas configuraciones a React, y se desarrolle el portlet según se configure.

En este caso se podría haber realizado una configuración en el portlet de Liferay para poder elegir el color de los gráficos, o la forma en la que queremos que se visualicen los gráficos. Estas configuraciones realizadas desde el componente en Liferay podrían haber cambiado dichos colores o dichas formas.

Otro posible trabajo futuro sería poder incluir *Redux*[1], que hace que se actualice automáticamente la aplicación si un componente cambia, lo que permite que se individualicen los problemas de una manera más rápida. Si tenemos una aplicación muy grande y se realiza un cambio en un componente una vez el proyecto está casi acabado, *Redux* hace que se actualice el cambio en la aplicación, al igual que sus variables sin necesidad de hacerlo manualmente.

Otra ventaja de *Redux* es que si se actualiza el estado de una variable en un componente se actualiza también en el resto de componentes del programa.

[1] `https://es.redux.js.org/`

Bibliografía

[1] I. K. Center. Conceptos de los portlets. https://www.ibm.com/support/knowledgecenter/es/SSYJ99_8.5.0/dev-portlet/wpsbpc.html.

[2] L. DXP. Plataforma de experiencia digital, 2019. https://www.liferay.com/es/products/dxp.

[3] M. Haverbeke. *Eloquent JavaScript, Third Edition*. No Starch Press, 2011.

[4] JSON. Introducing json. https://www.json.org/.

[5] Liferay. Introduction to liferay development, 2017. https://dev.liferay.com/es/develop/tutorials.

[6] Liferay. Liferay portal, 2017. https://dev.liferay.com/.

[7] Liferay. Portlets, 2017. https://dev.liferay.com/es/develop/tutorials/-/knowledge_base/7-0/portlets.

[8] McGraw-Hill. Sistemas gestores de contenidos. Technical report, McGraw-Hill. https://www.mheducation.es/bcv/guide/capitulo/8448183924.pdf.

[9] D. G. Miguel Ángel Álvarez, Miguel Ángel Durán. *Manual de React*. 2019.

[10] React. React - the anti-counterfeiting network, 2019.
https://www.react.org/.

[11] T. Soriano. ¿qué es un gestor de contenidos o cms y cuáles son los más usados?, Nov. 2016.
https://nocionesunidas.com/blog/tecnologia/gestor-contenidos-cms-cuales-los-mas-usados/#.XPaZYIjHw2w.

[12] J. J. Villar. *Descubre React*. 2016.

[13] J. X. Yuan. *Liferay Portal Enterprise Intranets*. PACKT Publishing, 2008.