UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Desarrollo de una aplicación web de visualización de eventos para los portales web de la Generalitat Valenciana.

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Carlos Sendra Gisbert

Tutor: Moisés Pastor Gadea

Curso 2020-2021

Resumen

Este proyecto consiste en el análisis, diseño y desarrollo de una aplicación web basada en la tecnología portlet del gestor de contenidos web Liferay dedicada a la visualización de eventos de cada uno de las páginas web de la GVA. El objetivo es actualizar la versión de la aplicación, la cual actualmente está basada en la versión 6.1 de Liferay, para que posteriormente se pueda migrar todos los portales a la nueva versión 7.2, que incluye varias mejoras tanto en la visualización como en el rendimiento de las páginas web.

El cliente es la propia Generalitat Valenciana, la cual ha delegado a Alfatec desde hace varios años la gestión y desarrollo de todos los portales y funcionalidad web de la mayoría de sus conserjerías. La aplicación que se nos ha encargado está dirigida a los usuarios de la GVA, es decir, cualquier persona que quiera información sobre un próximo evento en alguna conserjería podrá hacerlo entrando a la página pertinente y consultando nuestra aplicación.

Dado que la diferencia entre la versión actual de Liferay y a la que se pretende migrar es bastante considerable, se nos ha pedido realizar un estudio previo para incorporar nuevas funcionalidades que ofrezca el gestor de portales a nuestro programa, además de evitar cualquier tipo de conflicto en la base de datos existente.

El programa se desarrollará sobre la versión 7.2 de Liferay, con Java, JSP, Oracle <Database y Gradle en backend, y HTML5, CSS3 y ReactJS en el *frontend*.

Resum

Aquest projecte consisteix en l'anàlisi, disseny i desenvolupament d'una aplicació web basada en la tecnologia portlet del gestor de continguts web Liferay dedicada a la visualització d'esdeveniments de cadascuna de les pàgines web de la GVA. L'objectiu és actualitzar la versió de l'aplicació, la qual actualment està basada en la versió 6.1 de Liferay, perquè posteriorment es puga migrar tots els portals a la nova versió 7.2, que inclou diverses millores tant en la visualització com en el rendiment de les pàgines web.

El client és la mateixa Generalitat Valenciana, la qual ha delegat a Alfatec des de fa diversos anys la gestió i desenvolupament de tots els portals i funcionalitat web de la majoria de les seues conselleries. L'aplicació que se'ns ha encarregat està dirigida als usuaris de la GVA, és a dir, qualsevol persona que vulga informació sobre un pròxim esdeveniment en alguna conselleria podrà fer-ho entrant a la pàgina pertinent i consultant la nostra aplicació.

Atés que la diferència entre la versió actual de Liferay i a la qual es pretén migrar és bastant considerable, se'ns ha demanat realitzar un estudi previ per a incorporar noves funcionalitats que oferisca el gestor de portals al nostre programa, a més d'evitar qualsevol tipus de conflicte en la base de dades existent.

El programa es desenvoluparà sobre la versió 7.2 de Liferay, amb Java, JSP, Oracle Database i Gradle en back-end, i HTML5, CSS3 i ReactJS en el front-end.

Abstract

This project consists of the analysis, design and development of a web application based on the Liferay web content manager portlet technology dedicated to displaying events on each of the GVA web pages. The objective is to update the version of the application, which is currently based on Liferay version 6.1, so that later all portals can be migrated to the new version 7.2, which includes several improvements both in the visualization and in the performance of the webpages.

The client is the Generalitat Valenciana itself, which has delegated to Alfatec for several years the management and development of all the portals and web functionality of most of its ministries. The application that has been commissioned to us is directed to the users of the GVA, that is to say, anyone who wants information about an upcoming event in any department will be able to do so by entering the relevant page and consulting our application.

Given that the difference between the current version of Liferay and the one to be migrated is quite considerable, we have been asked to carry out a preliminary study to incorporate new functionalities offered by the portal manager to our program, in addition to avoiding any type of conflict in the existing database.

The program will be developed on Liferay version 7.2, with Java, JSP, Oracle Database and Gradle on the *backend*, and HTML5, CSS3 and ReactJS on the *frontend*.

Tabla de contenidos

Índice de figuras

1.Introducción

Cada día que pasa nuestra vida orbita un poco más alrededor de Internet. Los negocios ya se llevan a cabo en su mayoría de forma *on-line*, las citas del médico se solicitan mediante una aplicación web y no tienes ni que levantarte del sofá para que te traigan la cena a casa. En este contexto de digitalización, contar con una presencia en la red actualizada, accesible y profesional es indispensable para la satisfacción de los usuarios.

La necesidad de renovación de las páginas web de la Comunitat Valenciana es evidente desde hace mucho tiempo. Las páginas que se saturan con facilidad, con estilos antiguos y obsoletos, y los servidores que dejan de funcionar con frecuencia mellan en la paciencia de los ciudadanos de la Comunitat. El propio gestor de portales que utilizan las páginas de la Generalitat Valenciana, Liferay, hace años que actualizó su código fuente para proporcionar una experiencia más fluida y dinámica. Sin embargo, seguimos estancados con la vieja versión.

En esta memoria se detallará una pequeña aportación a este gran cambio que supone la actualización de los servidores reescribiendo desde cero el código de una aplicación de la Generalitat Valenciana a la última versión de Liferay.

1.1 Motivación

Como se ha hablado en la introducción, no solo es necesaria la presencia en Internet; también es indispensable mantenerse actualizado con las últimas tecnologías para obtener el mayor rendimiento y satisfacción de los usuarios. Una migración de los servidores de Generalitat no solo mejoraría la experiencia de aquellos que utilizan los portales, sino que es un soplo de aire fresco para los usuarios editores que pueden trabajar en un entorno con la última versión de uno de los gestores de páginas web más importante en su sector.

A nivel personal, afronto este proyecto con una gran ilusión, pues es uno de los más grandes en la empresa, y me servirá para aprender cómo se trabaja en un equipo, a utilizar herramientas profesionales y desarrollar con lenguajes y tecnologías que no conocía y que son indispensables para el éxito de mi futuro laboral.

1.2. Objetivos

El objetivo principal de este proyecto es desarrollar una aplicación que permita la visualización de eventos en los diferentes portales de la Generalitat Valenciana, desarrollada bajo la versión de Liferay 7.2. Este portlet debe, además, seguir visualizando sin errores los datos ya existentes en la base de datos en el momento del despliegue a los servidores.

Como objetivos secundarios, se pidió, entre otros:

- Reducir el tiempo y la carga de ejecución de los portales en los que se incluya.
- Facilitar a los usuarios el uso de este portlet.
- Reducir al mínimo el trabajo que deben realizar los usuarios para utilizar esta nueva versión.
- Cumplir con los requisitos de accesibilidad y diseño *responsive* en la web.

1.3. Impacto esperado

Esperamos desarrollar una aplicació más amigable y con una interfaz más cuidada, fácil de utilizar, tanto por los editores como por los usuarios, con un mejor rendimiento en los servidores. Todo esto sin que sea necesario alterar demasiado el procedimiento de creación y visualización de eventos y, en todo caso, generar un cambio positivo en el proceso de creación.

Con esta aplicación estamos un paso más cerca de la migración de los servidores de la Generalitat a la última versión de Liferay, un cambio que los ciudadanos de la Comunidad Valenciana agradecerán con creces, dadas las quejas actuales sobre nuestras páginas y su funcionamiento.

La migración del servidor afectará, obviamente, a todos los portales web de la Generalitat Valenciana pero, en concreto, el desarrollo de este portlet será relevante en todas las organizaciones que hospeden eventos, excursiones, charlas, o deban señalar fechas importantes. Algunos ejemplos de estas organizaciones pueden ser todos los portales de Parques Naturales de la GVA (www.parquesnaturales.gva.es), los cuales servirán de ejemplo en toda la memoria. Se espera también ampliar el uso de esta aplicación a portales como el Instituto Valenciano de Cultura, donde los eventos se añaden de forma manual con la tecnología HTML, un proceso largo y tedioso.

1.4. Metodología

Para la creación del portlet se ha seguido un desarrollo iterativo e incremental, donde se analiza, diseña y desarrolla una parte de la aplicación, se hace pruebas con ella y se vuelve al principio del ciclo, el cual acaba con el desarrollo de una nueva parte del portlet, hasta que se tiene la aplicación completa y funcionando. Esta metodología ha sido muy útil para contar con el *feedback* del cliente, ya que le fue presentado el portlet en varias ocasiones antes de la finalización del mismo.

En concreto, cada parte del ciclo ha sido:

- **Análisis**: En esta parte se estudia el estado de arte del portlet minicalendario en la versión 6.1,y nos reunimos con el cliente para ofrecer varias mejoras y descubrir las necesidades de los usuarios. Además, se elaboran documentos de toma de requisitos, que incluyen las necesidades específicas de los usuarios, sin entrar en los detalles de la implementación. Es este documento el que guía el trabajo de las siguientes etapas.
- **Diseño**: La fase del diseño compone el análisis de la solución y una creación de la estructura base, elección de las tecnologías de desarrollo, etc., para luego poder elegir una estrategia de desarrollo óptima.
- **Desarrollo**: Es en esta etapa donde se implementa todo lo que se diseñó en la fase anterior. Se utilizan las tecnologías propuestas y se solventan los errores de programación. En esta etapa estamos comunicándonos con el cliente de forma periódica para que pueda ver nuestro progreso y comentar sus dudas o inquietudes.
- **Verificación y pruebas**: Dado que la migración de los servidores es un proceso muy lejos de estar finalizado, no se puede probar el funcionamiento de la aplicación en un entorno de producción. Sin embargo, internamente, la GVA cuenta con un servidor Liferay 7.2 donde se puede probar sin riesgo de modificar datos relevantes de los portales. Una vez se verifica que todo lo que se quería implementar funciona correctamente, se presenta la parte del proyecto al cliente y se procede a la siguiente iteración del mismo.
- **Mantenimiento y mejoras**. Esta es una fase con vistas al futuro, pues este portlet no va a poder salir a la luz hasta que el servidor entegro sea migrado. Sin embargo, se irá manteniendo conforme nuevas versiones de Liferay aparezcan, como es el caso de la reciente versión 7.3. Estos trabajos de mantenimiento se detallarán en el apartado trabajos futuros de esta memoria.

1.4. Estructura

Esta memoria se compone de ocho capítulos, además de las referencias, glosarios y los anexos. Cada capítulo tiene diferentes apartados y subapartados. A continuación, se resumen brevemente los capítulos:

1. **Introducción:** Se introduce el tema a tratar en la memoria, la motivación y el impacto que se espera tener con la realización del proyecto, y se explica la metodología que se seguirá en el desarrollo del mismo.

2. **Situación actual:** Este capítulo intenta analizar el entorno en el cual se va a desarrollar la aplicación, para entender cuál es la magnitud del problema.

3. **Análisis del problema:** Siendo la continuación del anterior apartado, aquí se estudia el estado del arte de la aplicación minicalendario y de la herramienta Liferay, cuáles son las funciones que se deben mantener en la nueva versión, y se plantean diferentes soluciones al problema ya establecido.

4. **Diseño de la solución:** Antes de comenzar a escribir código es indispensable conocer cuáles son las partes de nuestra aplicación e intentar diseñar cómo se pueden desarrollar. Este proceso se cubre en el capítulo actual.

5. **Detalles de la implementación:** En este capítulo se describen los detalles del desarrollo, se documenta en mayor profundidad los problemas que han ido surgiendo a lo largo del mismo, y se explica también la solución a cada uno de ellos.

6. **Validación:** Se detalla cómo se ha probado el correcto funcionamiento de la aplicación, desde la compilación hasta el despliegue y ejecución en servidores oficiales.

7. **Conclusión:** En este apartado se recapitula lo que se ha hecho hasta ahora, incidiendo en qué se ha aprendido y qué puede servir para aplicarlo en un futuro en el ámbito laboral.

8. **Trabajos futuros:** Por último, se explica qué se ha quedado por hacer, cuál es el destino de nuestra aplicación o algunos problemas que aparecieron una vez finalizado el desarrollo.

1.5. Convenciones

Las palabras que no se acepten en la RAE, como los englicismos o conceptos pertenecientes a las tecnologías utilizadas se escribirán en cursiva.

Aquellas palabras o conceptos que pertenezcan a archivos o códigos del proyecto se escribirán en negrita.

2.Situación actual

El primer paso para la migración de una aplicación web debe de ser un análisis de la realidad, para tener una base clara de qué problema existe y cuál debe ser la mejor solución. Para un correcto análisis, debemos estudiar quién es el cliente, cuál es el contexto tecnológico y qué soluciones podemos ofrecer al problema planteado. Este análisis corresponde al capítulo actual de la memoria y al siguiente.

2.1. El cliente

La Generalitat Valenciana es el conjunto de instituciones de la Comunitat Valenciana. Forman parte de ella las Cortes Valencianas, el Presidente y el Consejo de la Generalitat Valenciana.

Como institución oficial del Estado, es indispensable que la GVA tenga una presencia en la web profesional, accesible y actualizada. Desde 2013[1] cuenta con el apoyo de Alfatec, ahora Lãberit, para proporcionar unos portales web acorde a las necesidades de los usuarios y trabajadores.

Por cada institución debe haber un portal web único y personalizado, dispuesto a suplir las necesidades individuales que cada conserjería pueda tener. Varias de estas consellerias cuentan con un calendario personalizado para mostrar los eventos, actividades y seminarios que tengan relevancia para tal organización.

2.2. Liferay

Liferay es un gestor de portales web de código abierto basado en la tecnología Java. Su objetivo principal es permitir a sus usuarios crear páginas web de manera modular, utilizando sus llamados portlets, aplicaciones que cumplen una función específica.[2] Así, una aplicación puede mostrar artículos y novedades de prensa, y otro puede buscar un alto cargo de la GVA de entre todos los existentes.

Además, permite la creación de propias aplicaciones para suplir las necesidades determinadas de cada usuario mediante su API y diversas herramientas que proporciona a los desarrolladores.

En este contexto de desarrollo propio entra el portlet minicalendario.

3.Análisis del problema

Para poder proponer una solución adecuada, es necesario conocer el estado actual del portlet minicalendario y cuáles son sus funciones, y estudiar los cambios que se introdujeron en la última versión de Liferay.

3.1. El portlet minicalendario

El portlet minicalendario fue desarrollado por Alfatec para la versión 6.1 de Liferay. Cumple la función de la visualización de eventos creados por un portlet externo, con una vista personalizada para cada organización o conserjería que lo utilice.

Cabe tener en cuenta que la aplicación minicalendario no se encarga de crear, modificar o eliminar eventos. Para realizar cualquiera de estas funciones, será necesario acceder al panel de control del portal y modificarlo desde ahí. Dicho funcionamiento es el mismo para la gestión de artículos, documentos y multimedia. Como alternativa, se puede utilizar la aplicación oficial del calendario de Liferay, la cual se puede enlazar desde nuestro portlet si así lo pide el cliente.

Actualmente el portlet minicalendario se puede encontrar, por ejemplo, en todos los portales de parques naturales de la Generalitat, siendo https://parquesnaturales.gva.es/va/ la página “padre”, y cada parque natural uno de sus hijos.

El minicalendario cuenta con tres maneras de visualizar los eventos: como una lista, con un formato mensual o como las dos a la vez. Para cada una de las maneras existen varias opciones de filtrado y presentación de dichos eventos.

Vista “Calendario”

La vista por defecto del portlet es el calendario del mes actual, con los días que tienen eventos destacando sobre el resto.

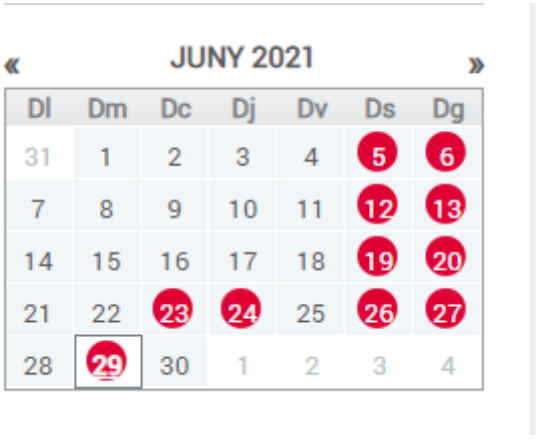

Figura 1: Vista "Calendario" por defecto.

Al pasar el ratón por encima de uno de ellos, aparece una pequeña caja de texto con un listado de los eventos de ese día. Si se clica sobre cualquier día resaltado, el portlet redirige al usuario a otra página donde se encuentra un segundo portlet minicalendario con la vista "Lista de eventos" activa.

Figura 2: Vista "Calendario" con un resumen de los eventos de un día.

El modo de comunicación entre un portlet y otro es mediante la construcción de una URL personalizada. En este enlace encontraremos que se especifica el día, el mes y el año que se quiere mostrar, además del identificador del portlet minicalendario destino.

Vista "Lista de eventos"

En esta vista se muestran los eventos de uno en uno, con su fecha de inicio, el título y un pequeño resumen. El resumen cuenta con una imagen pequeña y un texto descriptivo acerca del objetivo de dicho evento.

Los eventos están paginados, y se puede elegir cuantos eventos por página mostrar. En caso de que únicamente haya una página, no se muestran los controles para poder avanzar o retroceder entre los eventos.

Clicando en cualquiera de los eventos visibles se puede acceder al detalle del mismo, que se explica a continuación.

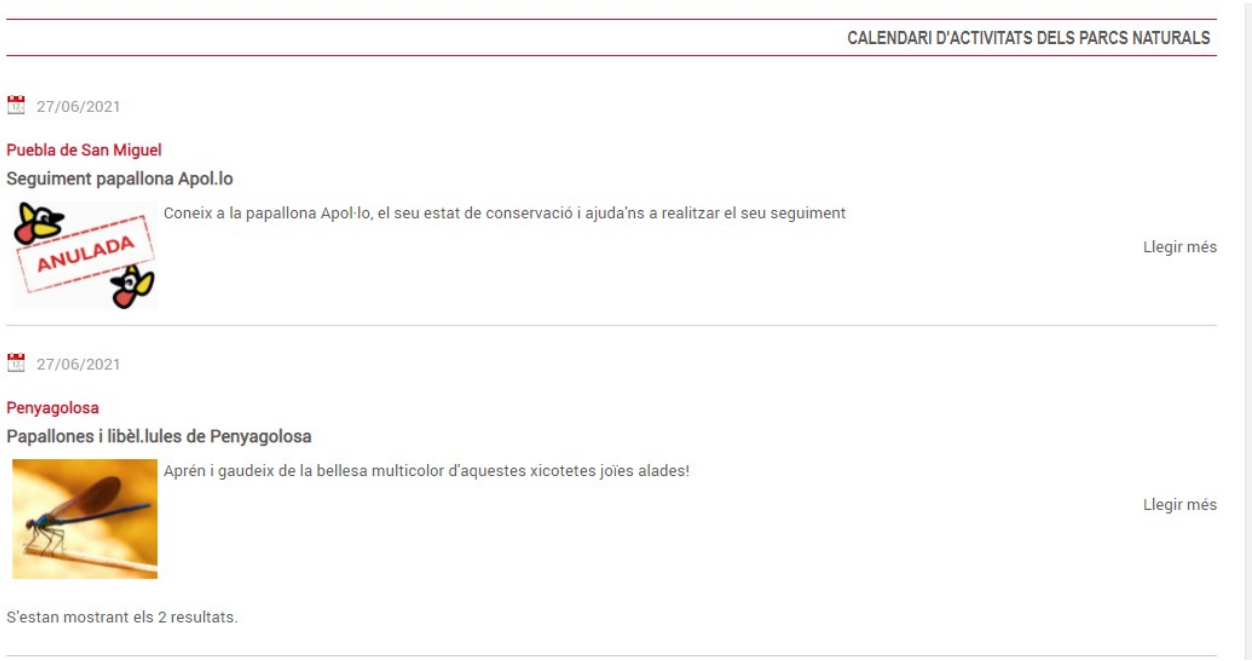

Figura 3: Vista "Lista de eventos".

Detalle de un evento

En el detalle de un evento se muestra el título y una descripción completa, la cual los usuarios editores tienen total libertad para mostrar lo que deseen, pues se puede incluir contenido HTML, como imágenes, estilos personalizados e incluso código JavaScript, aunque no suele ser el caso.

Figura 4: Vista del detalle de un evento.

El detalle de un evento deberá estar en, mínimo, castellano y valenciano obligatoriamente. Como veremos en el apartado de "Opciones de Presentación", este requerimiento genera un problema en cuanto a la creación de eventos con Liferay, pues no trae por defecto la funcionalidad de añadir una traducción en las descripciones.

Vista "Ambos"

Esta vista únicamente se compone de las dos vistas principales anteriores, "Calendario" y "Lista de eventos", con la particularidad de que el detalle de los eventos se puede mostrar en el mismo portlet. Actualmente esta vista se ha dejado de utilizar en los portales.

Opciones de presentación

Todas las instancias del portlet minicalendario pueden ser personalizadas a gusto de los usuarios editores de las organizaciones. A continuación resumimos las diferentes opciones de presentación y comportamiento que se pueden modificar. Para poder acceder a esta configuración únicamente se necesitará estar en el portal deseado y clicar en el icono de la llave inglesa que tiene cada portlet.

Figura 5: Cómo acceder a la configuración de la aplicación minicalendario.

La opción "**Comportamiento de los links de los días del calendario**" se podrá elegir qué pasa cuando se clica en un día.

Si se escoge la primera opción se mostrará la pestaña del día escogido. Es necesario que la página a la que se dirige contenga el portlet Agenda para su correcto funcionamiento.

Por el contrario, la segunda opción solo enlazará los días con eventos. Para que funcione, es necesario que la página que se enlace contenga el portlet Mini-calendario.

Por defecto, la página a la que enlazan los días es la página actual. Esto se puede cambiar desde la llave inglesa, Apariencia y "**Enlazar URLs del portlet con la página**"

La opción "**Calendario por defecto dentro de la página**" es útil cuando hay más de un portlet minicalendario en la página. Establecerá cuál es el portlet que muestre los eventos cuando se enlace desde otras páginas. En caso de haber más de un portlet con esta opción marcada como sí, se definirá por defecto la primera que se encuentre.

Por último, si se quiere añadir eventos desde el propio portlet, sin entrar en el panel de control, se podrá establecer una página del portal donde se alojará la aplicación oficial del calendario de Liferay. Esta aplicación queda fuera del ámbito de nuestra memoria, pero será necesario explicar su funcionamiento a los usuarios editores de los portales.

Filtrado de eventos

En el apartado filtro de eventos, podemos elegir cómo mostrar eventos, cuáles ocultar e incluso mostrar eventos de otras organizaciones en una organización padre.

Mostrar eventos pasados y futuros permitirán ocultar esos eventos, dependiendo del uso que se le quiera dar al portlet.

Mostrar eventos de las organizaciones hijas permitirá que las organizaciones padres de otras organizaciones muestren sus eventos como propios.

Se podrá también escoger de qué organizaciones mostrar los eventos, a partir de los **identificadores** de esas organizaciones. Para escoger qué organizaciones mostrar se deberá especificar de la siguiente manera: GroupId1, GroupId2...

Por último, se puede filtrar según coincida **alguna o todas las categorías** especificadas en los ajustes, o todos los eventos si no se escoge ninguna categoría.

Cambiar visualización del evento por la del contenido relacionado

Uno de los requisitos impuestos por la Generalitat Valenciana es que todos los contenidos deben estar en valenciano y castellano, y en ocasiones en inglés. Sin embargo, en la versión 6.1 de Liferay, los contenidos de un evento no son traducibles, por lo que se ideó una solución que involucra a los contenidos relacionados

Se puede sustituir el título del evento con un **contenido relacionado** que hayamos adjuntado al evento. Para ello, se ha de asignar una URL amigable que contenga el publicador de contenidos donde se muestran los mismos. Por último, se tiene que especificar el identificador del publicador dentro de esa URL. El identificador se encontrará en la opción “estilos avanzados”, dentro de la opción de apariencia del portlet publicador de contenidos. En caso de no existir contenido relacionado, se mostrará el título original sin ningún problema.

Para cada una de las vistas existen ciertas configuraciones específicas, que se detallan a continuación.

Configuración de la vista "Calendario"

- Se pueden mostrar en el calendario eventos según a qué **tipo** pertenecen. No se utiliza dado que es una opción por defecto, y es preferible filtrar por categorías, la opción de configuración del apartado anterior.
- Se pueden mostrar **categorías o el nombre de la organización** a la que pertenece el evento delante del título.

Configuración de la vista "Lista de Eventos"

- Se puede especificar un **título** para esta lista. Sólo es necesario si elige el tipo de visualización **Ambos**.
- La ordenación de los eventos puede ser **Ascendente** o **Descendente**.
- De nuevo, el tipo de los eventos es una opción que no se utiliza.
- Se puede mostrar una **lista por páginas**, estableciendo cuántos eventos se muestran por página. Para no paginar, se dejará en blanco o 0.
- Se puede mostrar la **hora** de los eventos, el **tipo**, la **imagen** asociada, etc.
- Si **no existen eventos**, se puede ocultar el portlet con la opción Mostrar portlet sin eventos.
- Por último, se puede especificar **cuántos eventos se mostrarán** en la lista. En blanco se mostrarán todos.

Configuración vista "Ambos"

- Esta configuración es complementaria de las dos anteriores.
- Se puede mostrar en **dos columnas** o el calendario encima de los eventos.
- Se puede elegir entre mostrar los **eventos próximos**, **de hoy** o **del mes** al entrar en la página.

Configuración de la apariencia

Desde configuración del portlet (la llave inglesa) también podemos configurar la apariencia del mismo. Ésta configuración será idéntica para las tres vistas del portlet. Pese a que hay más opciones, para el desarrollo del *portlet* solo se necesita la siguiente:

Enlazar URLs del portlet con la página: Redirige a la página una vez se clica en un día en la vista "Calendario". Por defecto es la misma página, pero puede cambiar a cualquiera dentro del ámbito del portal.

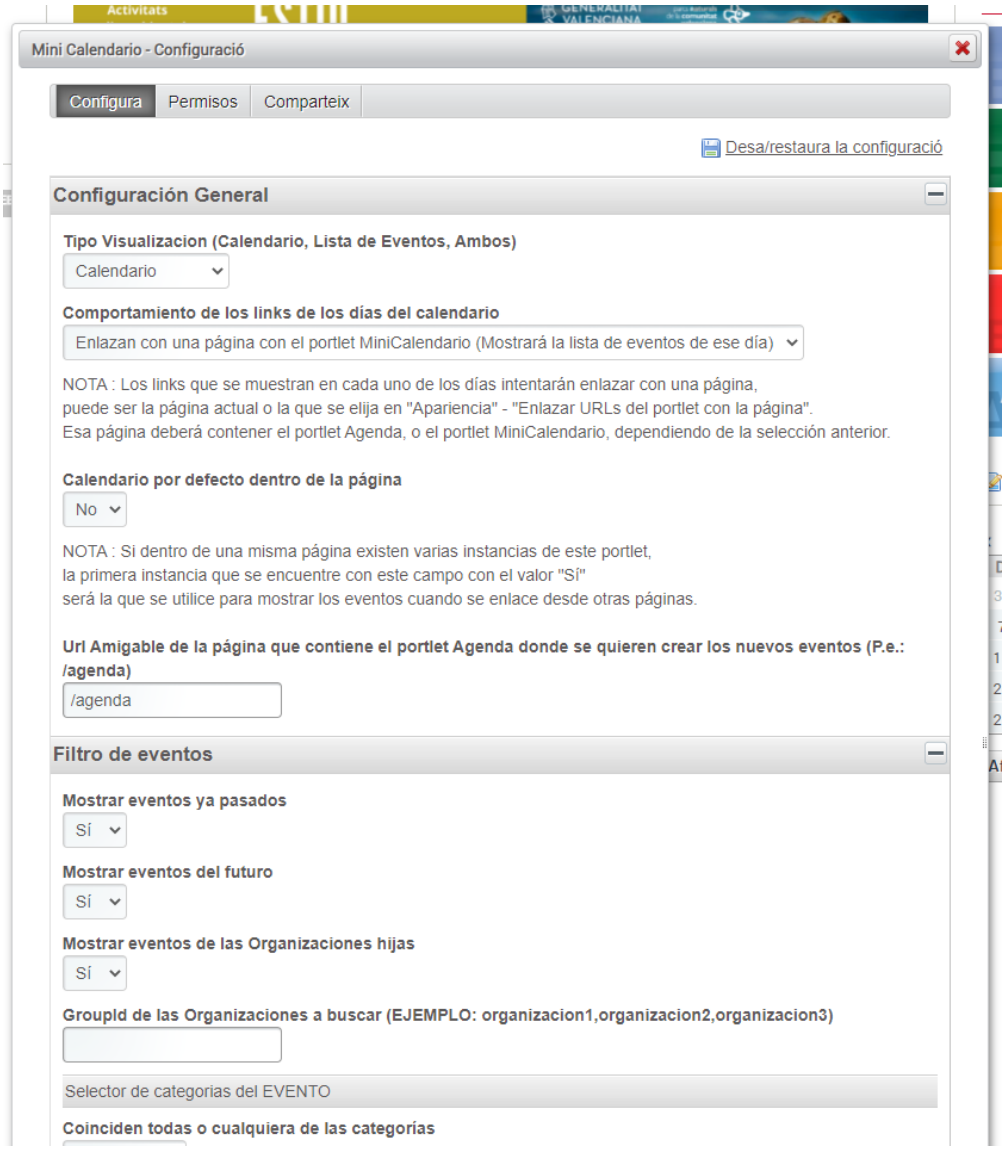

Figura 6: Ejemplo de cómo se ven las opciones de presentación del portlet minicalendario

3.2. Qué ha cambiado en la nueva versión

La diferencia entre la versión 7.2 de Liferay y la 6.1 es de, aproximadamente, ocho años. En todo ese tiempo, se lanzó Liferay 7.1 y, más importante, Liferay 7.0 o el inicio de Liferay DXP. En esta versión, se cambió completamente el paradigma de programación en el gestor de portales. A parte de actualizar el lenguaje Java a la versión 8, se incluyó soporte a OSGI y comenzó la modularización de su core.[3] Como veremos más adelante en los detalles de la implementación, esto nos ha permitido omitir varias etapas del desarrollo mediante la importación del código oficial de Liferay a nuestra aplicación.

En esta versión, enfoca el desarrollo más en el lado del cliente que en el servidor, permitiendo integrar frameworks de JavaScript a nuestros proyectos.

Por último, pero no menos importante, Liferay ha cambiado por completo la apariencia de su panel de control y páginas, además de añadir varias funcionalidades a sus aplicaciones.

Dentro de todos estos cambios, podemos encontrar varios que afectan de manera directa a nuestro portlet minicalendario. Por ejemplo:

- Los eventos de Liferay ya son traducibles en su totalidad, lo que deja obsoleta una funcionalidad base del minicalendario, los contenidos relacionados.
- La introducción de nuevos frameworks en JavaScript, como ReactJS, podrían mejorar el rendimiento del servidor, muy saturado en la actualidad.
- Varios métodos del portlet se han visto obsoletos o directamente dejan de funcionar en la nueva versión, lo cual fuerza, como mínimo, a modificarlos.

3.3. Análisis de soluciones

Con toda esta información, tenemos claro cuáles son los problemas de seguir manteniendo el portlet original en nuestro nuevo servidor, por lo que se debe buscar una solución óptima. A continuación se detallan las soluciones propuestas al cliente, y la explicación de por qué se escogió desarrollar la aplicación desde cero.

Calendario oficial de Liferay

Liferay ha desarrollado y mantiene activamente un calendario en sus servidores, tanto en 6.2 como en las últimas versiones. Es, de hecho, desde donde se crean los eventos que más tarde se obtienen en nuestro propio portlet. Sin embargo, se desestimó su

utilización en Liferay 6.2 por su falta de funcionalidad, que se suplió con el desarrollo de nuestro famoso minicalendario. Es ahora, con la nueva versión de la plataforma, cuando surge la duda de si esas funcionalidades ya se cubren, haciendo innecesaria la utilización de nuestro portlet.

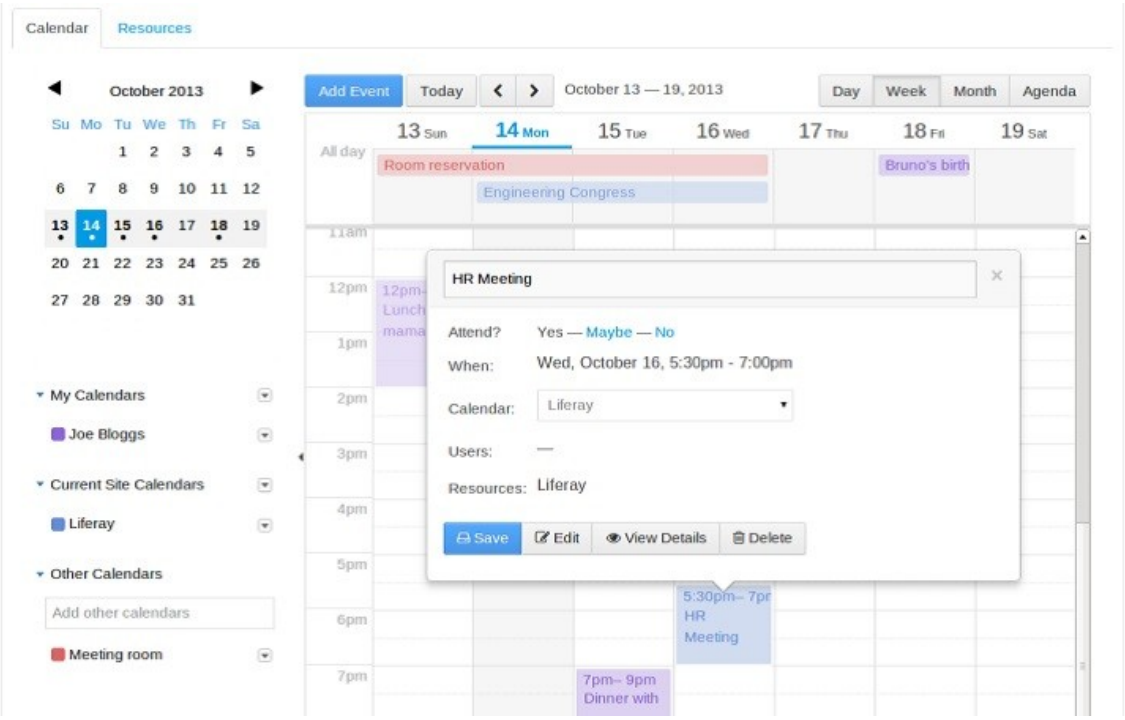

Figura 7: Calendario oficial de Liferay.

Si bien es cierto que ha crecido con los años y es extremadamente completo, sigue enfocándose en una intranet más que en un visualizador de eventos. Algunas de las funcionalidades que pide el cliente pero con las que el calendario de Liferay no cuenta son:

- El calendario de Liferay no tiene una vista única de calendario, sino que se ha de ver junto con un organizador de días, semanas o meses.
- No se permite adjuntar y visualizar una foto pequeña en la “Lista de eventos”, tal y como pretende el cliente.
- No se puede mostrar una pequeña descripción en el resumen de los eventos en la “Lista de eventos”.
- El filtrado de eventos no existe. No se puede mostrar eventos por categorías, o dejar de mostrar eventos pasados o futuros.
- La funcionalidad de pasar el ratón sobre los días con eventos de lo que sería la vista “Minicalendario” para ver un listado de eventos no existe.
- No se pueden visualizar los eventos de las organizaciones hijas.

Con todas estas carencias, se descartó en seguida la utilización de este calendario como sustituto del minicalendario.

Clonar el calendario oficial de Liferay

El calendario carece de funcionalidades, pero es una aplicación muy potente y completa. Se podría pensar que, al tener el código fuente del portlet, se podría copiar todo y cambiar la visualización para adecuarla a lo que le interesa al cliente.

En principio se intentó desarrollar el minicalendario de esta manera, pero nos dimos cuenta de que el portlet original tenía muchas más funcionalidades de las que necesitábamos, que ralentizaban la ejecución y nos dificultaba el desarrollo de código propio . Además, clonar el código generó muchos conflictos en la base de datos, y al final nos fue imposible desarrollar nada de valor.

Utilizar calendarios de terceros

También se podría estudiar si existen implementaciones de minicalendarios en la web, ya sea para Liferay u otras tecnologías.

Liferay tiene una especie de mercado donde los usuarios pueden poner a la venta sus portlets propios. En este *marketplace* nos encontramos el portlet *Events List*[4], proporcionado por el propio Liferay, pero nos encontramos con un desarrollo pobre, en el que solo hay una lista de eventos, faltando el resto de opciones de presentación.

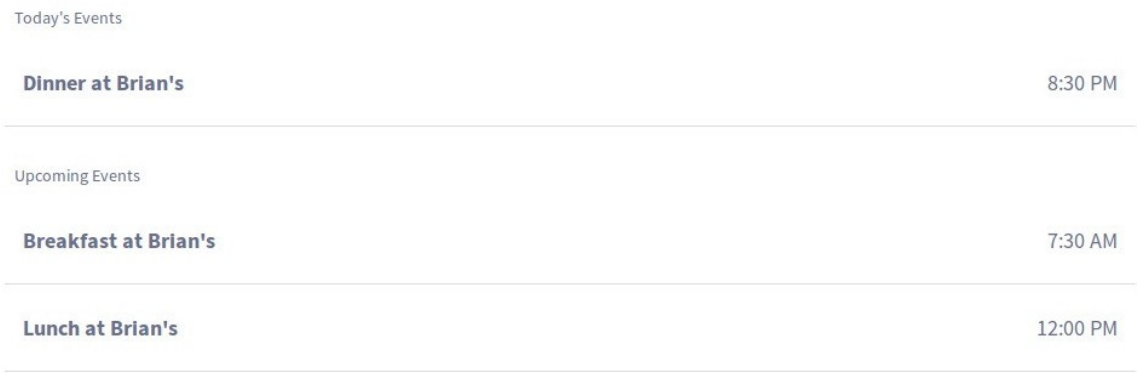

Figura 8: Liferay Events List

También se planteó como opción el portelt *Calendar Events Enhanced*[5], aunque tampoco cumple los requisitos del cliente y además no se puede modificar, dado que es de pago.

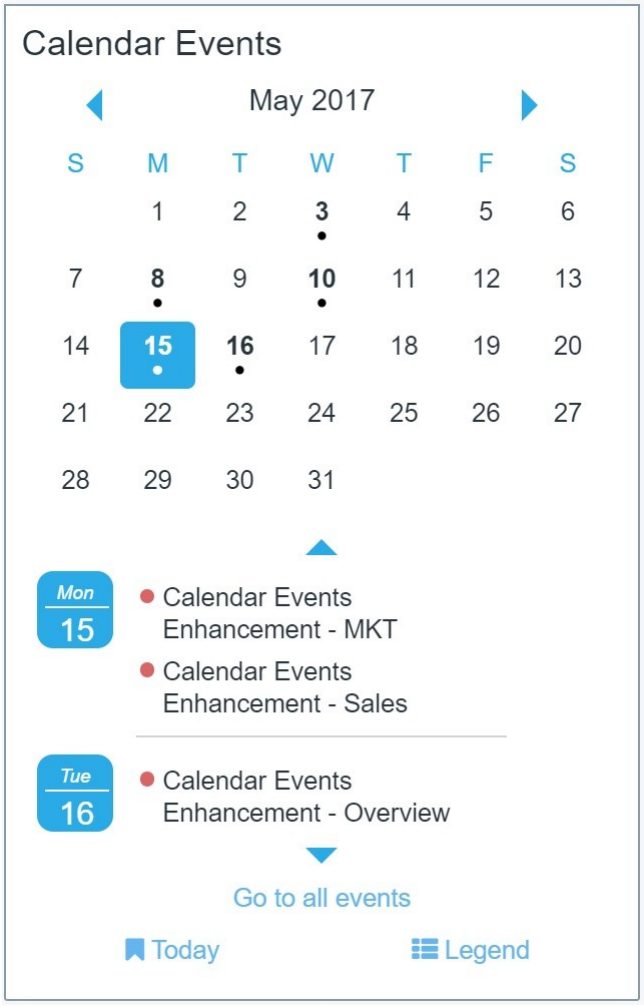

Figura 9: Calendar Events Enhanced

Por último, decidimos buscar no aplicaciones completas, sino módulos que se pudieran importar a nuestra aplicación con frameworks de JavaScript. Gracias a la gran comunidad de ReactJS y a su extensa modularización, encontramos *react-calendar*[6], un módulo de la vista "Calendario" altamente personalizable, que cumple con casi todos los requisitos de la propia vista.

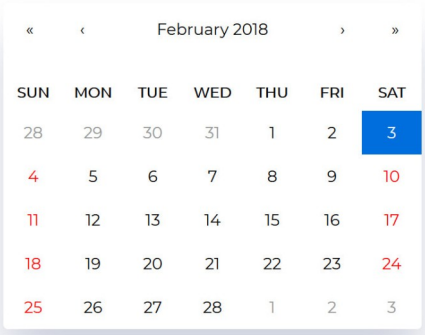

Figura 10: El módulo React Calendar.

Desarrollo del calendario desde cero

Ya que encontramos un calendario muy completo en forma de módulo de ReactJs, y gracias a que Liferay ya permite la integración de frameworks como este, la opción de crear un portlet desde cero se hacía cada vez más tangible.

Pese a que parece más difícil volver a crear el portlet desde cero, el módulo que podemos importar, sumado a la mejora de rendimiento que supone incorporar ejecución por el lado del cliente, decantó la balanza hacia esta solución.

3.4. Solución propuesta

Al final se decidió desarrollar el portlet desde cero porque, aunque parece más complicado, dadas las nuevas herramientas de desarrollo, el alivio que supone una implementación ya hecha de la vista "Calendario" y el mejor rendimiento que proporciona la integración de un framework JavaScript para gestionar la lógica de la visualización por el lado del cliente, es la mejor opción de las existentes.

En el diseño de la nueva aplicación se verá que algunas funcionalidades existentes en la versión anterior se desestiman debido a su falta de uso, como por ejemplo la vista "Ambos", y que se implementarán nuevas funciones y características, tanto pedidas por el cliente como de idea propia, mejorar el rendimiento y la presentación del portlet.

4.Diseño de la solución

En este apartado se detallan las diferentes etapas por las que debe pasar el desarrollo del portlet minicalendario, explicando la arquitectura del mismo, además de resumir las diferentes tecnologías que se utilizarán a lo largo de todo el proceso.

4.1. Creación del portlet

La manera más frecuente de desarrollar portlets para Liferay se inspira en la estructura MVC (Modelo, Vista, Controlador), la cual divide la aplicación en tres capas fundamentales:

La capa de modelo contiene todos los datos del portlet y la lógica para manipularlos. La capa de vista contiene la lógica para visualizar los datos. Por último, el controlador, la capa encargada de comunicarse e intercambiar datos entre las dos primeras.

Sin embargo, un portlet de Liferay no contiene las tres capas en sí misma, sino que opta por dividir su funcionamiento en diferentes módulos mediante una herramienta llamada *service-builder*[7].

Service-builder es una herramienta de creación de modelos XML para definir entidades que luego se convertirán en objetos en la base de datos, además de implementar métodos para conectar dicha base de datos con el propio portlet. Así pues, el Service Builder se encarga de crear un módulo *api* y un módulo *service* donde estará la parte de modelo del MVC.

El módulo *service* crea un XML con la estructura del campo que se quiera crear en la base de datos, y la interfaz necesaria para interactuar con dicha BBDD.

El módulo *api* crea los métodos que el controlador y la vista pueden usar para obtener los datos.

El portlet que crea el usuario es el que sirve de controlador y vista. Es esta aplicación el que queremos desarrollar nosotros, ya que vamos a usar los módulos *api* y *service* del calendario oficial de Liferay.

El portlet se creará mediante la herramienta que proporciona Liferay para el creado de módulos, temas y aplicaciones: Blade. Blade funciona por consola e incorpora varias plantillas para no tener que gestionar la creación de archivos y estructuras para el portlet. En concreto se utilizará la plantilla de npm-react, que permite la integración con la librería ReactJS.

4.2. Configuraciones

La configuración del portlet se hará conforme el estándar de Liferay, el modelo MVC. La creación de configuraciones en Liferay es relativamente sencilla, pues el propio gestor otorga facilidades para la comunicación entre la vista, el modelo y el controlador. Las configuraciones que se deben implementar se detallan a continuación:

- Elegir el tipo de vista: Entre "Minicalendario" y "Lista de eventos".
- Poder mostrar eventos pasados o futuros, o los dos a la vez.
- Mostrar las categorías a las que pertenecen los eventos.
- Mostrar la organización a la que pertenecen los eventos.
- Mostrar los eventos según las categorías a las que pertenezcan.
- Mostrar los eventos de las organizaciones hijas.
- Filtrar las organizaciones hijas de las que obtener eventos.
- En caso de la vista "Calendario", escoger a qué página se redirige al clicar encima de un día con eventos.
- En caso de la vista "Lista de eventos", ordenar de más reciente a más antiguo o viceversa, mostrar un número máximo de eventos, y escoger cuántos eventos aparecen por página.

4.3. Obtención de datos

Los datos se obtendrán mediante los métodos de los módulos service y api implementados por Liferay e importados en nuestro proyecto. Los datos que se deben obtener serán el calendario de la organización en la que se encuentre el portlet actual, los eventos de dicho calendario y los calendarios de las organizaciones hijas. Los eventos de las organizaciones hijas se deberán obtener si así se especifica en la configuración del minicalendario.

De los eventos que se obtengan, no toda la información es necesaria, y alguna requiere de un procesamiento extra, dado que se van a visualizar mediante JavaScript.

4.4. Procesamiento de datos

A la hora de obtener los eventos, se guardarán en una Lista de Java. Sin embargo, este objeto es incompatible con React o JavaScript, por lo que esta lista se deberá convertir en un objeto JSON para que se pueda leer correctamente.

También se crearán funciones en React para procesar datos que solo se puedan gestionar en tiempo de ejecución, como los eventos recurrentes, la paginación de los eventos, o el comportamiento del ratón.

4.5.Visualización de datos

La visualización de los datos y las diferentes vistas del minicalendario se conseguirán mediante la librería ReactJS, donde se crearán diferentes componentes para cada una de las vistas o fragmentos de las vistas. En concreto, la estructura de los componentes deberá quedar así:

- **Scheduler**. La vista "Lista de eventos". Desde aquí se hace un filtrado para mostrar los eventos correspondientes y se llama a otros componentes que gestionan una u otra lógica de la vista.
    - **Header**. En este componente se mostrarán la fecha de inicio, las categorías y la organización si procede.
    - **Event**. Desde aquí se visualiza el título y el resumen del evento. Cabe destacar que tanto este componente como Header se repetirán por cada evento que se deba mostrar.
    - **Pagination**. Este componente gestionará la lógica de la paginación de los eventos. Solo se mostrará una vez.
- **MiniCalendar**. La vista "Calendario". Es aquí donde se importará el módulo react-calendar que antes mencionamos, con algunas modificaciones para albergar todas las funcionalidades que nos pedía el cliente, como resaltar los días con eventos, o mostrar un pequeño resumen de los eventos en un día.
    - **TileContent**. Este es el componente que mostrará los eventos de un día a modo de resumen cuando se pase el ratón por encima de MiniCalendar
- **Modal**. Este componente es independiente del resto, pero solo se podrá acceder desde la vista "Lista de eventos". Muestra los detalles del evento, tales como la fecha de inicio y final, una descripción completa, etc.

4.6. Migración de datos

Ya se ha comentado que los eventos en Liferay 6.1 se basan en la visualización de los contenidos relacionados, y que esta es una funcionalidad que desaparece en el desarrollo de la aplicación en la nueva versión del servidor. Para poder seguir visualizando correctamente estos antiguos eventos, se deberá modificar en la base de datos ese contenido relacionado para que pase a formar parte del evento en cuestión, en forma de descripción y resumen.

Esta migración se realizará a través de la herramienta de ejecución de código del servidor de Liferay, donde se puede modificar la BBDD mediante el lenguaje Groovy y los diferentes servicios que ofrece la API del gestor de portales.

Además, ya que tenemos que desarrollar un portlet que requiera las mínimas modificaciones por parte de los usuarios, debemos analizar que, al desplegarse esta nueva versión, no haya cambiado la visualización en los portales. En caso de que haya riesgo de que no se vea igual que antes, se deberá modificar las configuraciones utlizando la misma herramienta que al migrar los datos, para migrar las configuraciones.

4.7. Tecnologías utilizadas

A continuación se describen las tecnologías y herramientas utilizadas tanto para la creación de la aplicación como para la comunicación con el cliente.

- **Liferay**. El gestor de portales web Liferay bajo el que podemos desplegar nuestra aplicación. Como hemos comentado en el capítulo Situación actual, el funcionamiento de Liferay se basa en aplicaciones o portlets modulares que conforman una página web completa. En la versión que vamos a utilizar desarrolla su tecnología en Java 8. A parte del servidor alojado en un dominio de la Generalitat, dispondremos de un servidor local para realizar pruebas más a menudo.
- **Blade**. Blade es una tecnología desarrollada por Liferay que asiste al desarrollador a crear un portlet con una estructura compatible con la plataforma.
- **Java**. Java es un lenguaje de programación distribuido actualmente por la empresa Oracle. Es el lenguaje que utilizaremos para desarrollar el back-end de la aplicación.
- **Oracle SQL**. La base de datos de Oracle es la que utiliza la Generalitat Valenciana para guardar toda la información de sus servidores. Accederemos a ella a través de la aplicación oficial de la empresa, SQLDeveloper.

- **ReactJS**. Esta biblioteca de código abierto de JavaScript es mantenida por la multinacional Facebook. Actualmente cuenta con más de mil desarrolladores libres que contribuyen a su desarrollo. Su funcionamiento consiste en la creación de componentes modulares que devuelven contenido HTML y JavaScript personalizado.
- **JSP**. Java Server Pages es una tecnologia que nos ayudará a desarrollar páginas web que se comuniquen con el *backend* de la aplicación mediante variables predefinidas por Liferay.
- **CSS**. Para definir los estilos de la aplicación utilizaremos el estándar de la web, CSS3.
- **Groovy**. Groovy es un lenguaje de programación orientado a objetos implementado sobre la plataforma Java. Este lenguaje es el escogido por Liferay para poder acceder a la base de datos desde su servidor, y poder realizar consultas mediante la API de Java.
- **Jira**. Para especificar los requisitos de la aplicación y comunicar el avance de la misma con el cliente utilizamos la plataforma Jira que se aloja en un dominio de la GVA.
- **Teams**. Microsoft Teams es la plataforma de comunicación escogida para comunicarse entre el equipo y con el cliente, para mostrar demos de la aplicación, recoger feedback.
- **Liferay Developer Studio y Visual Studio Code**. Para desarrollar la aplicación se utilizaron el IDE oficial de Liferay, basado en Eclipse, de Oracle, y el editor de textos de Microsoft.

5. Detalles de la implementación

En este capítulo de la memoria explicaremos en detalle las etapas de desarrollo de nuestra aplicación, así como las soluciones que aplicamos a cualquier problema que haya surgido.

5.1. Desarrollo del portlet

Como hemos mencionado en el diseño de la solución, este portlet se encarga de desarrollar las partes del controlador y visualización de eventos. Para crear la estructura base de la aplicación, utilizaremos la herramienta blade, que contiene plantillas predefinidas para cada una de las tecnologías. En concreto, dado que hemos decidido integrar la librería ReactJS en el proyecto, utilizaremos el siguiente comando por consola para crear una plantilla del portlet[8]:

```
blade create -t npm-react-portlet -v 7.2 [-p packageName] [-c className]
```

El argumento *-t* especifica la plantilla que se quiere utilizar. En concreto, npm-react-portlet es la plantilla que se utiliza cuando se quiere incluir cualquiera de los diferentes frameworks de Javascript, ya sea AngularJS, jQuery o, en este caso, ReactJS. La versión será 7.2, bajo la que queremos desarrollar el portlet.

Los argumentos *packageName* y *className* vienen dados por el portlet minicalendario de la versión 6.1, dado que se ha de llamar de la misma manera para que no haya conflictos en la etapa de la migracion. En concreto, *packageName* será **es.gva.minicalendario**, y *className,* **MiniCalendario**.

Una vez finalizada la ejecución del comando, se habrá creado una estructura de carpetas y archivos similar a la siguiente. A continuación se resume la funcionalidad de cada carpeta.

Figura 11: Estructura base de un portlet de Liferay

- La carpeta gradle contiene propiedades para la compilación y despliegue del portlet, y no es necesario modificarla.
- La carpeta en src se recoge el código fuente de la aplicación. En la subcarpeta main se divide ese código fuente en *backend*, java, y *frontend*, resources.
- En la carpeta **java** se recogen todos los archivos tipo java los cuales se encargarán de obtener los datos de la BBDD de Liferay, como los eventos, cuántos calendarios hay, gestión de permisos de visualización, etc.
- La carpeta **resources** engloba todos los archivos que corresponden con la visualización, como los archivos jsp, css, js, etc. En la carpeta content se añaden las traducciones a los idiomas que se requiera, en nuestro caso castellano, valenciano e inglés.
- Los archivos jsp son los que inician la visualización del portlet, ya que se llama desde la lógica de la aplicación. Se comunican con el *backend* mediante

variables predefinidas por Liferay, para posteriormente utilizar los datos que se puedan obtener para personalizar la presentación del portlet, mostrar diferentes eventos, etc. Liferay también proporciona diferentes etiquetas con las que se creará contenido HTML.

- En la carpeta **js** se incluirán los archivos pertenecientes a React. Se llamarán desde el archivo **view.jsp** y gestionarán la lógica de visualización completa, así como algunas funciones que puedan procesar datos de los eventos en tiempo real.

- Es importante discernir entre los dos tipos de archivos de visualización. Los archivos basados en Java (JSP) se ejecutan en el lado del servidor, por lo que no se pueden modificar en tiempo real. Sin embargo, los archivos React (JS), sí que se ejecutan en el lado del cliente. Esta distinción es indispensable tenerla en cuenta, ya que puede dar lugar a conflictos de ejecución. Por ejemplo, se pueden transferir datos desde JSP a React, pero no al revés.

- Los archivos que se encuentran fuera de la carpeta **src** establecen las dependencias del portlet, y lo definen dentro del entorno del servidor.

- Utilizaremos **build.gradle** para incluir las dependencias de Java, y **package.json** para las de React.

Veremos cómo esta estructura base se amplía con el paso del tiempo para añadir más carpetas y archivos, pero la idea principal no se modificará.

5.2. Gestión de dependencias

Para utilizar los servicios que proporciona Liferay, tales como el servicio del calendario, diferentes utilidades como el core de Liferay, se tendrán que importar al proyecto utilizando la tecnología Maven. Se deberá definir la inclusión de estos proyectos en el archivo build.gradle, especificando el número de versión. En concreto, estas versiones deben coincidir con las existentes en el servidor, por lo que deberemos consultarlas previamente. Además, es en este archivo donde pediremos a la aplicación que descargue *node*, la dependencia más importante de React. En caso de que *node* exista en el servidor, se saltará este paso.

En el archivo package.json se descargan React y todas sus dependencias, además de módulos que hayamos importado explícitamente para este calendario, como por ejemplo el módulo **react-calendar**, el cual ya hemos hablado de él en el análisis del problema. Se especifica dónde se encuentra el código de React, y cómo se va a llamar. Y, por último, se especifica el comando con el que se convertirán los archivos React a JavaScript normal, para poder ejecutar el código correctamente.

5.3. Inicialización del portlet

Ya hemos creado nuestra aplicación y hemos importado todos los módulos que necesitamos para su correcto funcionamiento. Sin embargo, aún no hemos definido nuestro portlet dentro del servidor. Para ello usaremos el tag *@Component* de OSGI. En este tag podremos especificar a qué categoría de portlets pertenece, dónde se encuentran los archivos de estilo o cuál será la visualización inicial que tiene que mostrar el portlet cuando se añade a una página web.

```
@Component(
    immediate = true,
    property = {
        "com.liferay.portlet.display-category=GVA",
        "com.liferay.portlet.header-portlet-css=/css/index.css",
        "com.liferay.portlet.instanceable=true",
        "javax.portlet.init-param.template-path=/",
        "javax.portlet.init-param.view-template=/view.jsp",
        "javax.portlet.name=" + MiniCalendarioPortletKeys.MINICALENDARIO,
        "javax.portlet.resource-bundle=content.Language",
        "javax.portlet.security-role-ref=power-user,user"
    },
    service = Portlet.class
)
```

Figura 12: Propiedades del portlet para identificarlo en el servidor.

La propiedad más importante es **javax.portlet.name**, que es el identificador de nuestro portlet. Como se puede apreciar, se encuentra en el archivo **MiniCalendarioPortletKeys.java**, el cual encapsula todas las constantes que vayamos a necesitar. En este caso particular, dado que es una migración de un portlet que ya existe en el servidor, el nombre tiene que ser el mismo que el antiguo, siendo este ***"minicalendario_WAR_minicalendarioportlet"***.

La categoría del portlet significa en qué pestaña se encontrará el portlet una vez desplegado en el servidor.

El resto de propiedades se crean por defecto con la plantilla que hemos utilizado de Blade, y no se requieren de modificación.

Con este paso finalizado, se podría desplegar sin problemas a un servidor de Liferay, aunque no se mostraría por pantalla nada más que un "Hola Mundo" genérico.

5.4. Obtención de los datos

Para poder visualizar los eventos correctamente, primero se tienen que obtenero.

El portlet, nada más se despliega ejecuta dos métodos iniciales, render y doView. En render se gestiona toda la lógica de obtención de datos y se añade a parámetros que luego se pueden pedir desde los archivos JSP, y doView importa los módulos JS y los incluye en una variable predefinida para poder obtenerse en la visualización.

Para poder consultar los datos de la BBDD, será necesario invocar los servicios de Liferay, aquellos que se crearon con Service Builder y estamos importando a nuestro proyecto.[9]

Los servicios de Liferay proporcionan una clase de tipo ***LocalService**, la cual es la que debemos importar y almacenar en una variable. Por convención, invocar el servicio de un calendario sería de esta manera:

```
@Reference
private CalendarLocalService _calendarLocalService;
```

La etiqueta **@Reference** que encontramos antes de la variable indica al portlet que estamos importando un servicio de Liferay. Con esta variable inicializada, podremos usar los métodos desarrollados en la API.

Obtener los eventos de Liferay no es una tarea tan fácil como parece. Si miramos en la documentación de Liferay, el método **getCalendarBookings**(long calendarId) devuelve todos los eventos de un calendario en concreto, el cual pasamos el identificador como parámetro. Es cierto que se pueden obtener todos los eventos de la BBDD, pero eso consume mucho tiempo cuando solo se pide los de una organización, que puede ser muy nueva y no tener ninguno o muy pocos.

Por lo tanto, es necesario conocer qué calendario es el indicado para nuestro portlet en cada una de las instancias.

Para obtener el calendario, la documentación no es de mucha ayuda, pues la lógica que usa Liferay internamente depende de variables que en nuestro portlet no existen. Es por ello que se necesita idear otra estrategia.

Contamos con la ventaja de que en todas las organizaciones existe un calendario por defecto, y se puede obtener de una manera relativamente fácil. Necesitaremos únicamente el identificador de la organización, que se obtiene con una variable general de Liferay, **themeDisplay**, la cual contiene información del servidor y sus organizaciones, así como de la página actual.

Este método devolverá el calendario por defecto, con el que podemos consultar su identificador. La única desventaja es que, si una organización crea más de un calendario, sólo se podrán obtener los eventos del calendario predeterminado. Sin embargo, el cliente está al tanto del tema y no ve ningún problema con únicamente utilizar un calendario por organización. Además, no hay antecendentes de que esto haya pasado en otros portales en la versión 6.1.

Si no ha habido ningún problema con la obtención del calendario, se añadirá como atributo a la variable **portletRequest**, que es el puente entre los archivos JSP y los de tipo Java. Como no nos interesa sólo el identificador del calendario, añadiremos el objeto entero como atributo.

Con el calendario de la organización conocido, se pueden obtener sus eventos. Los eventos internamente tienen cuatro estados: Aprobado, Pendiente, Quizás y Rechazado. Debemos especificar que ha de tener los tres primeros estados, ya que si se rechaza puede ser que los editores de la página lo hayan querido ocultar y no se atrevan a borrarlo.

El último paso es obtener los calendarios de organizaciones hijas, que pueden o no ser requeridas más adelante. Dado que no siempre se van a mostrar los eventos, únicamente se obtienen los calendarios de dichas organizaciones. En caso de que se pida visualizar sus eventos, se hallará la manera de obtenerlos más adelante. El tiempo de ejecución de la obtención de los calendarios de las organizaciones es trivial, puesto que la organización con más hijos no supera las diez organizaciones. Sin embargo, un solo

calendario puede contener miles de eventos, por lo que consumiría mucho más tiempo obtenerlos todos cada vez que se actualiza una página.

El último paso de la obtención de datos, y el último método de esta clase, es el llamado **doView()**. En este método se debe invocar la variable **_npmResolver** creada anteriormente que, de un modo similar a los servicios que hemos invocado anteriormente, permite crear una referencia del portlet, para luego poder añadir una etiqueta de Liferay en los archivos JSP donde se invoque dicho módulo. Sin el desarrollo de este método, no se podrían visualizar los archivos React que se hayan añadido a la aplicación.

```
@Override
public void doView(RenderRequest renderRequest, RenderResponse renderResponse)
    throws IOException, PortletException {
        renderRequest.setAttribute(MiniCalendarioConfiguration.class.getName(), _configuration);
        renderRequest.setAttribute(
                "mainRequire",
                _npmResolver.resolveModuleName("MiniCalendario") + " as main");
        super.doView(renderRequest, renderResponse);
}
```

Figura 13: Inicialización del minicalendario como módulo NPM.

5.5. Procesamiento de los datos

Se ha establecido en el diseño de la solución que las listas Java, que es el objeto con el que obtenemos los eventos de los calendarios, son incompatibles en JavaScript, por lo que es necesario convertir la lista en un objeto JSON.

Para solventar esta dificultad, se ha desarrollado un método que realiza la conversión de una manera personalizada. Así, se devolverá un objeto JSON en el que cada clave sea el nombre de uno de los diferentes calendarios, el principal y sus organizaciones hijas, si procede, y su valor sea un segundo objeto JSON, que tiene un día con eventos como clave, y los diferentes eventos de la misma fecha como valor.

El proceso será pues, recorrer la lista de calendarios que se piden en la visualización, y por cada uno de ellos añadir su título como clave de un objeto JSON. Acto seguido, llamar a un segundo método auxiliar que cree ese segundo objeto JSON. Como sabemos que los eventos en la lista de Java están ordenados por fecha, de más antiguo a más

reciente, solo hará falta recorrerlos desde el principio hasta el final e ir añadiendolos a las claves correspondientes. Así pues, por cada evento:

- Se incluyen el identificador del evento, el del calendario, el nombre del calendario, el título del evento y la descripción.
- Se obtendrá la recurrencia del evento, en caso de que tenga.
- Se obtendrán las categorías por las que esté clasificado.
- Se incluirá la fecha de inicio y la duración del evento. Esto se consigue restando la fecha de final con la del inicio. Explicaremos en el apartado de la recurrencia por qué es mejor incluir la duración que la fecha de final.
- Se incluye el resumen. El resumen de un evento no existe como tal en Liferay, sino que lo hemos creado nosotros a modo de campo personalizado. Los campos personalizados son una herramienta que proporciona Liferay para individualizar aún más las páginas web. De una manera simplificada, el resumen es un atributo de la clase Liferay ExpandoBridge. Este campo está en formato XML y se puede completar su contenido en varios idiomas. Sin embargo, para facilitar la visualización en JavaScript, se transformará en un objeto JSON.

Una vez acabado de incluir atributos en el evento, se comprueba si la fecha de inicio coincide con alguna clave del array de días. En caso positivo, se añade al array. Por el contrario, si es el primer evento del día, se crea un nuevo array y una nueva key.

5.6. Configuración del portlet

Este paso podría realizarse antes o después de visualizar los datos, pero dado que dicha visualización depende de parámetros y opciones de presentación, se ha decidido desarrollar antes que las diferentes vistas de la aplicación.

Para crear la configuración de Liferay es necesario crear una interfaz de configuración, con métodos para cada parámetro, un valor por defecto y especificar si se requiere para el funcionamiento o no.

Con el modelo creado, necesitamos controlar cómo actualizar los parámetros en la BBDD, y eso lo conseguimos gracias a la ventana de configuración personalizable que incluyen todos los portlets de Liferay, que tiene una clase específica, **MiniCalendarioConfigurationAction.java**. Contiene un método, **processAction**, que se activa cuando un usuario clica el botón guardar dentro de la ventana de

configuración. Lo único que se tiene que hacer para guardar los valores de cada parámetro es actualizar la variable **portletPreferences** con una clave y un valor. Esta clave será el nombre de la opción particular, y el valor dependerá del parámetro. Para actualizar la variable únicamente será necesario llamar al método **setPreference**, predefinido en el portlet.

Sin embargo, el usuario aún no puede actualizar ningún parámetro porque no se ha creado un campo en el formulario de configuración. Para ello, nos dirigiremos a **META-INF/resources/configuration**, donde existen varios archivos JSPF. Estos archivos son fragmentos que se pueden incrustar en una página JSP mayor, como es el caso de **configuration.jsp.** Dependiendo del parámetro de configuración, se deberá añadir en un formulario o en otro.

En este formulario se añadirá un campo acorde con el tipo de parámetro que se quiere implementar. Por ejemplo, si el valor del parámetro es una bandera booleana, se escribirá una etiqueta Liferay de tipo *checkbox*. Si es una lista con una opción, deberá ser una etiqueta del tipo *select*. Estas etiquetas, sin embargo, siempre tendrán un nombre, el cual debe coincidir con el nombre definido en el modelo del primer paso, y un valor por defecto. Este valor por defecto es el último paso en la creación de la configuración.

Por último, se necesita inicializar todos los parámetros en el archivo **init.jsp**. La incialización se completará obteniendo el valor del parámetro en cuestión por medio de la variable **portletPreferences**, a la que hemos añadido todas las configuraciones.

5.7. Inicialización de los datos

Init.jsp

Init.jsp es el archivo que se incluye en todos los archivos JSP que se crean en la aplicación, y tiene como funcionalidad obtener todos los datos relevantes en la visualización e importar las clases Java necesarias. En el caso del minicalendario, la segunda funcionalidad no es tan útil, porque no basamos el funcionamiento de la aplicación en arvhi, pero es igual de importante inicializar los datos y configuraciones.

Las acciones que se realizan en este archivo son la importación de clases útiles para procesar los datos, la inicialización de las opciones de configuración y el procesado de ciertas variables, como por ejemplo si incluir las organizaciones hijas a una lista de calendarios, que posteriormente se convertirán en el objeto JSON antes mencionado.

Por último, se requiere hacer ciertas comprobaciones para cada portlet, como por ejemplo construir las URLs que redirigen al calendario oficial de la organización a la que pertenecen.

Una funcionalidad curiosa es la de que las configuraciones booleanas en 6.1 eran cadenas de texto con valor "S" o "N", para determinar verdadero o falso. Sin embargo, y pese a todo lo que hagamos en DXP, los campos booleanos siempre se van a convertir a uno de los valores booleanos cuando se actualice la configuración en pantalla. Por tanto, se tiene que tener en cuenta un paso extra en ciertos parámetros, donde estará marcado como verdadero si es "S"o si es "true", y como falso en caso contrario.

View.jsp

El último paso antes de poder desarrollar las vistas del portlet es incluir en el archivo **view.jsp** un bloque HTML donde añadiremos los componentes de React. Este bloque tendrá el nombre del portlet como identificador.

Para poder añadir los componentes de React es necesario incluir en el archivo una etiqueta de tipo **aui:script**, con el atributo ***require*** inicializado al atributo que habíamos definido en el método **doView()** anteriormente. En esta etiqueta se llamará al método main que se encuentra dentro del archivo **Index.es.js**, el inicio del desarrollo en la biblioteca de JavaScrip. En esta función **main** añadiremos todos los parámetros que coinciden con las opciones de configuración. También será en este momento donde se llamará al método de conversión desde listas de tipo Java a objetos JSON.

```
<div id="<portlet:namespace />-root"></div>

<aui:script require="<%= mainRequire %>">
    main.default(
    '<portlet:namespace />-root',
    ...
    );
</aui:script>
```

Figura 14: Llamada al método main del archivo Index.es.js.

Index.es.js

Index.es.js es el archivo de inicialización de React. En él se encuentra la función principal que, dependiendo de los parámetros que se le haya pasado, decide qué vista mostrar y con qué opciones de presentación.

Lo primero que realiza el método es comprobar qué tipo de vista va a renderizar el portlet. Para ello toma máxima prioridad el detalle del evento. Es decir, si en la URL del sitio están los parámetros correctos, y el identificador del portlet coincide con el que está en la URL, ignorará el resto de parámetros, como por ejemplo el que dice que la visualización debe de ser una lista, y muestra los detalles del evento en concreto.

Detalles del evento

Si el portlet quiere mostrar esta vista deberá procesar los datos que tiene de calendarios para encontrar el evento en el objeto JSON. Para ello, lo primero que tiene que hacer es obtener la lista de eventos que tiene el calendario cuyo identificador se ha obtenido desde la URL, y encontrar los eventos que haya en el día determinado. Si no existe ninguno, puede ser por tres cosas. O la URL hace referencia a un evento que se ha eliminado de la base de datos, el enlace está mal escrito, o el evento tiene una regla de recurrencia.

La recurrencia se explica en su apartado corresponciente, pero por ahora basta con saber que se puede llamar a una función para intentar encontrar ese evento entre los eventos recurrentes.
Por último, independientemente del caso anterior, se busca con la función **.find()** el evento que tenga el mismo identificador que el obtenido en la URL, y se llama al componente React **Modal**, con el evento requerido y la fecha de inicio como propiedades.

Este es el primer objeto de React que nos vamos a encontrar. Antes de explorar sus detalles, es conveniente entender cómo funcionan estos componentes.

Un componente React dispone de un archivo .js único, por lo que siempre que hablamos de uno de ellos, se entiende que existe un archivo con el mismo nombre. Dentro de él, encontramos una clase que se extiende de **React.Component**, además de importaciones necesarias para cada archivo. Una clase de React funciona como cualquier clase de un lenguaje de programación orientado a objetos: contiene un constructor, y diferentes métodos. El método que devuelve contenido HTML, que es al final el objetivo de estos componentes, se llama **render()**, y se ejecutará en el momento en el que se llame la clase.

Por último, hay que tener en cuenta las propiedades y los estados de React[10]:

- Las propiedades son los parámetros que se pasan al componente una vez se llama desde otro archivo, como pueden ser en este caso, el evento a mostrar sus

detalles y la fecha de inicio. Internamente se accede a estas propiedades mediante el comando **this.props.propiedad**. Estas propiedades son inmutables, es decir, no pueden ser modificadas.

- Los estados son almacenes de datos mutables y autónomos por cada componente. Al poder ser modificados, son muy útiles para llevar la cuenta de variables que puedan influir en la visualización del componente. Una modificación de cualquiera de los estados fuerza la ejecución del método **render**.

Una vez asentadas las bases de las componentes, se puede continuar con la explicación.

Un componente **Modal** debe mostrar:

- El título del evento.
- El nombre de la organización a la que pertenecen.
- La fecha de inicio y el final. Estas fechas se deben mostrar en formato extendido p.e. Lunes, 06 de Marzo de 2021. Para mostrar esto en diferentes idiomas se deberá recurrir a la herramienta de localización de Liferay, que se explica en el apartado correspondiente a la localización..
- La descripción del evento en formato HTML, para que se pueda mostrar con estilos dependiendo de la intención de los editores.
- Si tiene, mostrar sus categorías.
- Si es recurrente, mostrar la regla de recurrencia. Esta regla de recurrencia se explicará en el apartado de la recurrencia.

En caso de que la url tenga un parámetro incorrecto, o el evento se haya borrado pero se intente visualizar, el objeto event será undefined en JS, es aquí donde se soluciona,

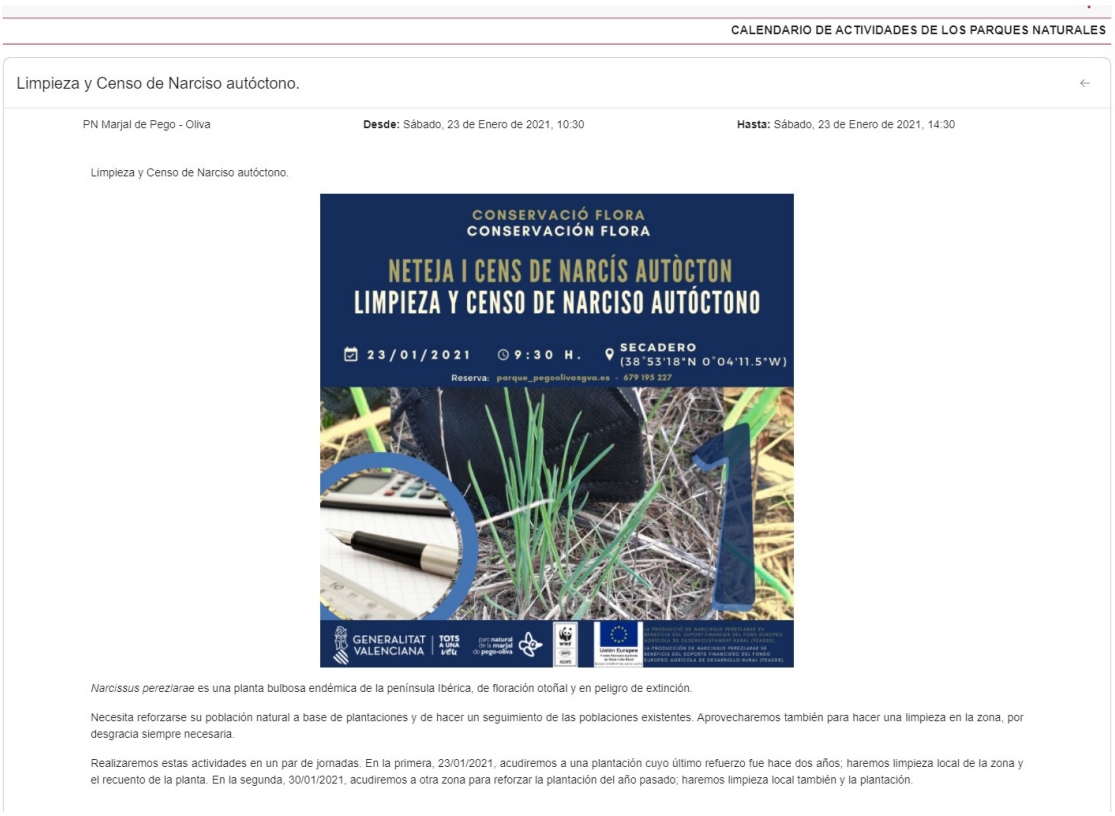

Figura 15: Detalle de un evento en la nueva versión.

Aquí finaliza el componente Modal, y pasamos a la siguiente opción de visualización, la vista “Lista de eventos”.

Lista de eventos

De vuelta en el archivo **Index.es.js**, la siguiente visualización posible, si no se debe mostrar el detalle de un evento, es la vista “Lista de eventos”. Se tomará esta opción en caso de que la variable **tipoVisualización** sea una cadena de texto con valor **L**.

Lo primero que debe hacer el portlet en esta vista es buscar eventos recurrentes y añadirlos a la lista de eventos en caso de que exista alguno.

En esta visualización también puede haber parámetros en la URL, pero son diferentes a los que existen en el componente **Modal**. En este caso, los parámetros sirven para mostrar únicamente la lista de eventos de una fecha en concreto. Esta URL nueva tendrá como parámetros el día, mes y año de la fecha indicada. No se consideró necesario incluir el identificador del portlet en este caso, pero se considerará en un futuro en caso de que así lo requiera el cliente en algún momento.

En caso de que la URL se considere válida, se buscarán todos los eventos que tengan la fecha de inicio indicada, a lo largo de todos los calendarios de la lista. En caso de que la URL no sea válida, o no haya parámetros en ella, se renderiza el componente **Scheduler** con los parámetros iniciales.

Scheduler es como se ha llamado al componente que gestiona la vista "Lista de eventos", y es el componente al que se llama en este momento.

Lo primero que se hace al crear este componente de React es inicializar el constructor, donde definiremos los siguientes estados:

- **todayDate**. el día actual.
- **orderBy**. El orden en el que se mostrarán los eventos, definido por la propiedad **order**. En caso de que el parámetro de configuración **ordenEventos** sea "asc", el valor de **orderBy** será 1. En caso contrario, será -1.
- **currentPage**. La página actual de eventos en la que se encuentra el paginador. Por defecto, al principio de la ejecución siempre será uno.
- **eventsPerPage**. Eventos a mostrar en cada página, definido por la propiedad **maxEvents**, la cual hereda el valor de la opción de configuración **maxDaysDisplayed**.

Lo siguiente es desarrollar la función que se va a ejecutar por defecto dentro de un componente, siendo siempre **render()**. Es en el retorno de dicha función donde se tiene que devolver contenido HTML para luego ser procesado. Así pues, para que **Scheduler** muestre información hay determinar qué eventos deben aparecer en el portlet. Para ello se requieren varias fases.

Los eventos se tienen que ordenar para mostrarse, comparando las fechas de inicio de mayor a menor o viceversa, dependiendo del parámetro **orderBy**, multiplicando por 1 si el parámetro de configuración es ascendente, y por -1 en caso contrario, invirtiendo el orden de la lista. Ya que el valor de **orderBy** puede ser 1, puede haber casos en los que en principio no deba requerir una ordenación. Aun así, contamos con la desventaja de que un objeto JSON es, por definición, una lista desordenada de eventos. Pese a que hayan sido ordenados previamente, no podemos suponer que lo vayan a estar en tiempo de ejecución. Por tanto, el objeto JSON que contiene los eventos se convertirá en un array de JavaScript en el momento de la ordenación.

Una vez se han ordenado todos los eventos, llega el momento de decidir si deben ser mostrados por pantalla o no cumplen con las condiciones necesarias. Para que un evento sea *renderizado* debe cumplir que:

- No hay un máximo de días mostrados o no ha llegado al máximo.
- No hay un máximo de eventos mostrados o no ha llegado al máximo.
- Se muestran los eventos pasados y es pasado o se muestran los futuros y es futuro.
- En caso de ser un evento del día actual siempre se mostrará.

Un evento que cumple estas características se añade a una lista nueva, cuyos elementos tienen un formato tal que:

```
{"calendar": Nombre_Calendario, "day": dd/mm/yyyy, "item": JSONevent}
```

Con los datos obtenidos, se tiene que preparar la paginación de los eventos.

Los pasos para mostrar eventos paginados es conseguir los índices del último y primer índice de eventos.

const **indexOfLastEvent** = this.state.currentPage * this.state.eventsPerPage;

const **indexOfFirstEvent** = indexOfLastEvent - this.state.eventsPerPage;

Vemos que el índice del último evento será el número de eventos por página por la página actual, p.e. 1 página * 3 eventos, el último índice, excluyéndose, es 3. El primer evento será este último índice menos los eventos por página: 3 - 3 = 0, incluyéndose.

Es en este momento en el que empezamos a pensar en estructurar el contenido HTML a mostrar. Aún no se va a enseñar nada, pero sí se va a preparar.

Dentro del *return* de la función, crearemos un bloque HTML con dos partes: la parte de los eventos y la parte de paginación. La parte de los eventos viene dada por una función externa llamada **renderData**(currentEvents), y la de paginación es la llamada a un componente React de desarrollo propio, llamado Pagination.

La función **renderData()** realiza unas últimas gestiones antes de devolver el contenido HTML. Por ejemplo, se debe comprobar que la lista proporcionada tiene eventos. En caso contrario, se debe devolver un bloque en el que ponga que no hay eventos, u

ocultar el portlet directamente. Esto se consigue añadiendo un identificador al bloque en cuestión, el cual consiste en el identificador del portlet más la cadena de texto **"_no-visibility"** como sufijo. Más adelante, en el archivo **view.jsp** hay un bloque de código JavaScript en el que oculta el bloque en caso de encontrar el identificador y el usuario que visite la página no está identificado como editor. En caso de que el usuario sí que sea editor le cambiará la opacidad para mostrar que existe, pero que no se mostrará por pantalla.

Si hay eventos en la lista, sin embargo, se requiere algo más de trabajo. Primero, se deben construir dos variables, que son la hora de inicio y las categorías a las que pertenece. Luego, se recorrerán los eventos de uno en uno y se llamarán a dos componentes más por cada uno de ellos: **Header** y **Event**, de los cuales el primero mostrará una pequeña cabecera del evento, con las categorías a las que pertenece, la fecha de inicio, y la organización, siempre y cuando se haya especificado en los parámetros de configuración que se enseñen, y **Event** mostrará el título y un resumen, consistente de una imagen pequeña y un texto.

El resumen del componente **event** es algo más complicado que un bloque de texto y quizá requiera una explicación extra. Se ha explicado que se debería haber creado un campo personalizado en el objeto Evento de Liferay oficial que describa el evento en cuestión. Sin embargo, y dado que el despliegue de la aplicación puede retrasarse un tiempo, no se controla si se creó e incluso puede directamente no rellenarse para todos los eventos, lo cual no es una situación improbable. En el caso de que el campo personalizado no exista, se ha decidido obtener los 300 primeros caracteres de la descripción como alternativa, para quizá tener una idea del objetivo del evento o, al menos, no dejar el campo en blanco. Aun así, puede ocurrir lo mismo con la descripción. Esto es, que no se haya rellenado, por lo que no nos queda otra solución que escribir que el usuario no ha introducido ninguna descripción para el evento.

No obstante, el componente Event no solo muestra información. También debe redirigir a los detalles del evento que esté mostrando. Para ello, en los campos de título, resumen y un botón que reza "Leer más", se crea un enlace a la url del evento. Esta URL se debe crear como ya hemos visto anteriormente, con las claves primarias **calendarId;** el identificador del calendario, **calendarBookingId;** el identificador del evento, la fecha del evento y el identificador del portlet donde queremos que se muestre. El motivo por el cual se incluye la fecha es por si acaso el evento es recurrente, ya que Liferay gestiona la recurrencia a partir de reglas, no creando un evento por cada día que ocurra.

La manera de crear una URL es mediante el objeto URL y su atributo **searchParams** de JavaScript, obteniendo el enlace original y añadiendo claves y valores.

```
let domain = window.location.origin + window.location.pathname;
let url = new URL(domain)
url.searchParams.append("calendarId", calendarId);
url.searchParams.append("calendarBookingId", calendarBookingId);
url.searchParams.append("date", date.replaceAll("/", "-"));
url.searchParams.append("portletId", this.props.portletId);
window.open(url.href, "_top");
```

Figura 16: Construcción de una URL personalizada con objetos JavaScript.

Con esto acaba el renderizado de los eventos en la lista, quedando por explicar sólo la paginación.

El componente **Pagination** cumple la función de mostrar el número eventos por página que se haya especificado en los parámetros. Para ello, se debe determinar que el estado de Pagination ha de ser el mismo que el de su padre, heredando sus propiedades. Esto es útil para saber en qué página de los eventos se encuentra y, sobre todo, para poder actualizar los estados.

El componente consta únicamente de cuatro enlaces y un indicador de la página en la que se encuentra. Los enlaces son Primero, Anterior, Siguiente y Último, que se encargan de navegar por las diferentes páginas de eventos. Cada uno de estos enlaces actualiza el estado de la página actual, restando o sumando uno, con un mínimo de 1 y un máximo de páginas disponibles de eventos. La página actual se puede modificar, ya que es una lista. Si se modifica también se va a la página que se haya clicado.

Como punto a clarificar, una funcionalidad que cambia del portlet de la versión 6.1 al desarrollo propio, es que la paginación siempre aparece, aunque únicamente haya una página de eventos. Esto es así debido a que varios usuarios experimentaban confusión a la hora de navegar por el portlet, debido a que este componente aparecía y desaparecía sin aparente criterio. No es una funcionalidad que el cliente pidió pero, al comentarle los problemas de los usuarios, aprobó esta característica.

En este punto, es conveniente recordar que actualizar el estado de un componente fuerza al mismo a volver a ejecutar su método **render()**. Por tanto, al actualizar el state de **Pagination**, heredado de su padre, podemos volver a relanzar el render de Scheduler con los nuevos estados. Esto quiere decir que se volverán a calcular los índices de los datos gracias a **this.state.currentPage**, y se mostrará la siguiente o anterior página de eventos, según proceda.

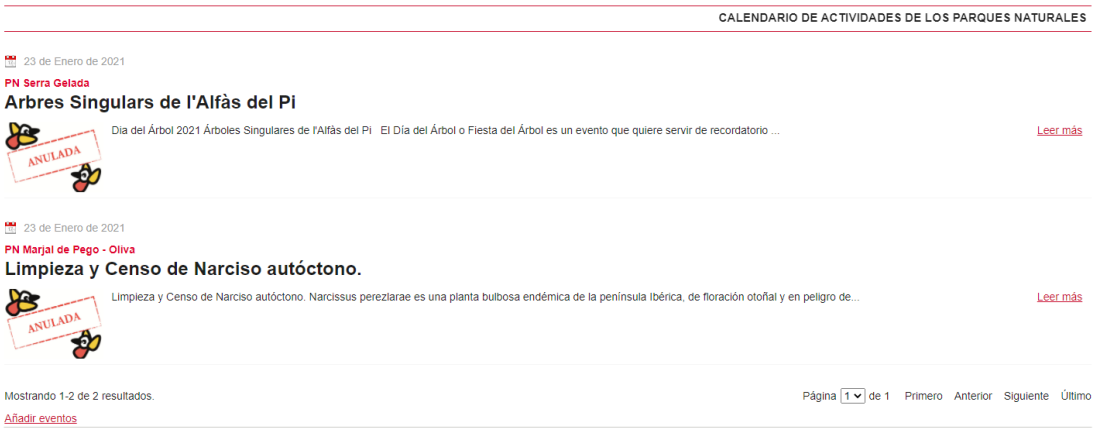

Figura 17: Vista "Lista de eventos" del desarrollo propio.

Vista Calendario

En caso de que el valor de "**tipoVisualización**" sea cualquier otro valor a "**L**", la vista que se renderizará será la de "Calendario".

Antes de poder llamar al componente, se deberán obtener los eventos recurrentes, si hay, de la misma forma que en la vista "Lista de eventos". En el único caso en el que no se llama de forma incondicional es cuando se encuentra el evento al mostrar los detalles de un evento, para ahorrar tiempo de computación.

El componente **MiniCalendar** es una importación de un módulo desarrollado anteriormente y publicado en el repositiorio de www.npmjs.com, el cual cumplía con la mayoría de funcionalidades pedidas por el cliente, y además nos permitía la personalización del módulo para desarrollar las que faltaban. Entre estas personalizaciones, se puede incluir contenido HTML extra, añadir clases siguiendo unas normas que han de cumplir los eventos, etc.

El componente calendario tiene la ventaja de que se puede navegar entre meses sin refrescar la página en la que se encuentra. Esta manera de mostrar datos obliga a tener los eventos cargados en el portlet, aunque no se muestren, ralentizando un poco la carga de la página. Sin embargo, compensa con la velocidad que se ha conseguido navegando entre meses para buscar eventos, frente a la lentitud del portlet original, que debía cargar la página por cada mes que se seleccionaba.

Las funcionalidades que se han desarrollado sobre el módulo importado han sido:

- Añadir una clase CSS en las casillas que contienen eventos, para más adelante destacar los días.
- Añadir contenido HTML en cada casilla con evento, mostrando una lista de los títulos de cada evento.
- Añadir un comportamiento extra cuando se clica un día con eventos, que consiste en redirigir al usuario a una página del portal con un portlet minicalendario en la vista "Lista de eventos", y mostrar el día seleccionado.

Las condiciones para destacar un día y mostrar los eventos que contenga son similares a los de la vista "Lista de eventos", aunque algo más sencillas. En concreto, para que un evento se muestre, debe cumplir que sea futuro y se puedan ver los eventos futuros, o que sea pasado y se puedan ver eventos pasados.

Figura 18: Vista Calendario de desarrollo propio.

El contenido HTML que se crea por cada casilla con eventos es un componente más, llamado **TileContent**. Este componente muestra como cabecera el día actual, en formato de fecha descriptivo. Por ejemplo, para el 20/06/20 la fecha a mostrar sería 20 de Junio de 2020. El componente toma en consideración si en las preferencias de presentación de eventos se ha marcado que se muestre el nombre de la organización y, si no es así, especifica en los estilos del bloque **display:none;**.

A continuación, por cada evento que se le haya pasado desde **MiniCalendar**, mostrará sus categorías, si procede, y su título.

Por último, cada vez que el usuario hace *click* en cualquier día con evento, se deberá construir una URL de una manera similar a la vista en el componente **Event**, aunque esta vez los parámetros serán día, mes y año, y se obtendrán de la fecha de la casilla seleccionada mediante un método de desarrollo propio.

Figura 19: Vista Calendario con un resumen de los eventos de un día en concreto.

La vista “Calendario” es la última de nuestro portlet. Sin embargo, los mayores problemas surgen a la hora de procesar los datos, no de mostrarlos. A continuación se detallan estos problemas, con su debida solución.

5.8. Recurrencia

Cuando un evento va a ocurrir más de una vez, quizá toda una semana, o cada primero de mes de un año, lo ideal no es crear un objeto por cada día que ocurra. Una posible solución es crear una regla que diga que un evento se va a repetir los lunes y miércoles, cada semana, durante los próximos 5 meses. Esta solución es lo que se llama una regla de recurrencia, y es como Liferay y la mayoría de calendarios gestionan estos casos de eventos repetibles.

La recurrencia en Liferay puede ser:

- Diaria.
- Semanal.
- Mensual, por día de la semana o día del mes.
- Anual, por día de la semana o día del mes.

Con un intervalo determinado, que indica cada cuántos días, semanas, meses o años se debe repetir.

Y puede acabar:

- Nunca.
- Después de un número determinado de veces.
- Al llegar a un día concreto en el calendario.

La regla en sí misma es una cadena de texto, que para el ejemplo anterior sería:

FREQ=WEEKLY;**UNTIL**=20211219;**INTERVAL**=1;**BYDAY**=MO,WE;

Liferay permite cambiar un evento, o una serie de eventos, en caso de que ese día ocurra una situación especial, o se deba modificar alguno de los datos. En estas situaciones sí que se crea una instancia nueva para el evento y, aunque se borra la regla de recurrencia de sus datos, se sigue considerando recurrente. Para que no genere conflictos en un futuro esta situación especial deberá ser tratada al procesar la recurrencia. En caso de que exista una excepción como estas, se añadirá a la regla de recurrencia general un nuevo campo de nombre EXCEPTIONDATE, con la fecha del evento anómalo como valor.

FREQ=WEEKLY;**UNTIL**=20211219;**INTERVAL**=1;**BYDAY**=MO,WE;**EXCEPTIONDATE**=20220407;

Pero la recurrencia es un problema a la hora de mostrar eventos por pantalla, porque solo tenemos un objeto con reglas, y necesitamos mostrar todos los eventos uno a uno. Así que hay que decodificar esta regla y añadir los eventos recurrentes en la lista que ya obtiene el resto de objetos, de forma ordenada. Esta decodificación se encuentra en el archivo **util.js.** El método **renderRecurring()** es el que busca todos los eventos recurrentes en la lista de eventos que se le pase, y crea copias del mismo, pero en la nueva fecha calculada.

Si encuentra una regla de recurrencia, lo primero que ha de hacer es convertir esa cadena en algo que se pueda leer de una manera más fácil: un objeto JSON con la parte izquierda del igual como claves y la derecha como valor, y cada regla separada por el punto y coma.

A continuación se iniciará un bucle por cada recurrencia para crear todos los eventos pertinentes, así que debemos saber cuándo pararlo. En caso de que la opción de finalización sea nunca, podemos establecer un número arbitrario, por ejemplo 99 ocurrencias, el cual es un límite muy elevado. En caso de que el fin sea una fecha y no un número determinado de ocurrencias se deberá añadir una condición al final de cada iteración del bucle, para que el nuevo evento no se escriba más allá de la fecha límite.

A continuación, obtenemos la fecha de inicio de nuestro primer evento. Este día es la base para el resto de cálculos del bucle.

Ahora bien, la manera de crear nuevas instancias es diferente según sea día, semana, mes o año.

En caso de que sea diario

En este caso el cálculo es sencillo, se suma un día multiplicado por el intervalo al día base. Esto es, si es diario con un intervalo de uno, el evento se repite cada día. En caso de que el intervalo sea dos, el evento se repetirá cada dos.

En caso de que sea semanal

Esta opción es bastante más complicada que la anterior, porque al poder tener múltiples días en sus reglas, la suma de días no es constante, y puede ser que se añada más de un evento por iteración del bucle, excediendo los posibles límites.

Además, un evento dado puede empezar un jueves, pero se repite cada lunes, martes y jueves, por ejemplo. Es importante no añadir un evento el lunes de la semana inicial,

porque sería incorrecto. Un inconveniente añadido es que los días que se repite el evento puede que no estén ordenados, lo que hace este proceso algo más complicado, pues no se pueden recorrer los días en orden ascendente.

Para crear eventos se deberá empezar con un primer bucle, que empieza en el día del evento y acaba en el día siete o domingo.

Si entre esos días se tiene que crear un evento, se obtiene el índice del día en cuestión, y el índice del día del evento y se calcula la siguiente fórmula:

let **sumaDias** = (7 - (*actualDay* - *i*)) % 7 || 7;

El || es por si es 0, se tienen que sumar 7 días también.

A continuación se tomará en consideración el intervalo, de manera que sumamos lo que haya dado la fórmula más 7 * (el intervalo - 1).

Se añade el evento y en el caso de que se pueda seguir creando eventos, el bucle realiza la siguiente iteración hasta el final.

Como en todos los casos, se obtiene como día de referencia el del último evento, y se vuelve a recorrer un bucle pero empezando desde el lunes hasta el día del último evento, repitiendo los pasos del bucle anterior.

En caso de que sea mensual o anual

En estos dos casos, lo único que cambia es sumar al mes o al año, pero el procedimiento es el mismo.

Aquí hay una condición extra, el día de la semana o del mes. Repetir el día del mes significa que el número 27 es cuando se realiza el evento, independientemente del día de la semana. Esto funciona igual que el caso de que la recurrencia sea diaria, se suma 1 mes o año multiplicado por el intervalo correspondiente a la fecha actual.

En caso de que sea el día de la semana hay un valor añadido en las reglas, en el que **BYDAY=1TH**, quiere decir el primer jueves del mes. En caso de que queramos el último, pondrá **BYDAY=-1TH.**

Para conseguir colocar bien el evento sin conocer qué día será el último jueves de mes, por ejemplo, se deben realizar los siguientes paso:

- Se crea una nueva fecha con el mes y el año correcto, pero en el primer día.
- Se calcula la diferencia entre el día de la semana y el que se quiere conseguir. Si esa diferencia es menor a 0, se suma 7, porque es el primer día.

- Si piden el 1, 2, o 3 semana del mes, se suma la diferencia de días y luego se suma el número de semanas que pidan, 7*0, 7*1 o 7*2.
- Si piden la última semana, se suma pero con 4 semanas. Esto puede no ser suficiente, pues hay meses con 5 semanas, pero no con 6. El siguiente paso es comprobar cuántos días quedan para el final del mes. Si la diferencia es mayor o igual a 7, quiere decir que no hemos llegado al último día de la semana, así que se añaden 7 días más a la fecha objetivo.

Una vez comprobada qué recurrencia tiene el evento, y cuál es la siguiente fecha a añadir en nuestra lista, debemos crear una copia y comprobar que entra dentro de los límites. En caso de que la frecuencia haya sido semanal, ya se habrán añadido todos los eventos, pero si no, se tiene que añadir ahora el del bucle que corresponda. Es en este momento donde se debe comprobar si se puede añadir el evento o se debe parar el bucle.

Si se deben añadir eventos hasta alcanzar un cierto número de ocurrencias, no hace falta comprobar nada porque las ocurrencias son la propia condición del bucle, y solo se necesitará restar uno al contador. Si es hasta una fecha concreta, se comprueba que la nueva fecha sea menor o igual a la límite, y si es así, se añade a la lista.

La función de añadir un evento solo crea una copia del anterior, le cambia la fecha de inicio y lo añade en el array que hay que devolver, en la posición correspondiente. Es por esto que se añade la duración del evento y no la fecha de fin, porque tendría un coste de computación mayor.

De regla de recurrencia a una cadena de texto

La recurrencia se debe mostrar en los detalles del evento, pero ya hemos visto que está escrita de una manera un tanto críptica para los usuarios, de modo que se desarrolló una función que traduce estas reglas a un lenguaje un poco más comprensible. Así, para la regla de ejemplo en el principio de este apartado, se leería por pantalla:

Repetir cada semana, hasta el 19 de Diciembre de 2021, los Lunes y Miércoles.

5.8. Localización

Como estamos en la Comunidad Valenciana, tenemos 3 idiomas en los que poner los portales, castellano, valenciano e inglés. Liferay proporciona una herramienta de localización muy buena, que es el objeto JS Liferay, que tiene dentro Language, que puede coger claves de idioma y lanzar su valor por pantalla, o en su defecto, escribir la clave. Por ejemplo, para la palabra lunes, se deberá escribir **Liferay.Language.get("lunes")** y, dependiendo del idioma en el que esté el portlet en ese momento, escribirá Lunes, Dilluns o Monday.

Estos valores se configuran por defecto en archivos **.properties** que se encuentran dentro de la carpeta **resources/content**. Aquí, podremos crear un archivo por cada idioma, llamándolo **Language_es.properties**, por ejemplo, para el idioma en castellano.

Su sintaxis es muy sencilla, consiste de una clave, un igual y un valor. En las claves no puede haber espacios pero sí puntos y guiones, y en los valores puede haber todo lo que esté codificado en **utf-8**.

5.9. Estilos

En cuanto a los estilos del minicalendario, se ha intentado asemejar lo máximo posible a la versión anterior del portlet, pero manteniendo las nuevas guías de diseño de la versión 7.2 de Liferay. Se ha cumplido con los requisitos de accesibilidad del cliente y el diseño *responsive* que se pedía, para su correcto funcionamiento en dispositivos móviles.

Sin embargo, dado que esta aplicación abarca múltiples portales, cada uno con un estilo diferente, se delega la personalización del portlet a las organizaciones convenientes, por medio de desarrollo de CSS propio, o mediante la utilización de temas predesarrollados por el propio equipo Alfatec, que será la situación más frecuente.

5.10. Migración de datos

Los eventos en 6.2 ya hemos comentado que no se podían traducir, por lo que se creaba un contenido web, que sí se traducía, y se relacionaba al evento. De esta manera, se obtenía una descripción traducible, pero era un proceso tedioso y complicado, ya que se creaban dos contenidos web para solo un evento, además de generar una carga en el servidor innecesaria. Con la entrada de Liferay DXP los eventos sí que se pueden traducir, así que se pidió desechar la idea de los contenidos relacionados. De cara a futuros eventos, no hay problema porque basta con crearlos, pero son los eventos

pasados los que han dado algo más de trabajo, ya que no se verán bien en el nuevo portlet a no ser que se pase ese contenido web a un campo ya existente de los eventos.
Además, se descartó la idea de dejarlos como están, pues muchos de esos eventos son relativamente recientes, están programados para un futuro cercano, e incluso son importantes aun siendo eventos pasados. Por tanto, la solución era modificar los eventos desde la base de datos, mediante la ejecución de código groovy en Administración del Servidor de Liferay, del panel de control. Groovy es un lenguaje basado en Java, que permite comunicarse con la base de datos desde el propio servidor, y realizar modificaciones mediante métodos implementados por el propio Liferay.

Hay tres cosas que deben ser migradas de los contenidos relacionados: la imagen pequeña, el contenido y el resumen.
La manera que se tiene de encontrar los eventos con contenidos relacionados en la base de datos no es por el propio evento, sino por el campo AssetLink, que enlaza un contenido con un evento. Se obtienen todos los AssetLinks en los cuales uno de los pares sea un evento. Para cada evento con contenido relacionado se guardan dos entradas en la base de datos, lo cual es un problema como veremos a continuacion.

Una vez encontrados los enlaces necesarios, y la comprobación de que no había ningún error en ellos, podemos empezar con la migración. Lo primero que se debe hacer es obtener el evento del calendario mediante los atributos de uno de los pares del AssetLink, y el artículo que corresponde al contenido relacionado del evento, un objeto JournalArticle. Es importante tener en cuenta que los artículos en Liferay tienen diferentes versiones y se pueden aprobar o cancelar, por lo que es importante obtener siempre la última versión aprobada.

A continuación se crea una lista vacía para ir añadiendo los diferentes idiomas en los que pueda estar escrita el resumen, ya que al ser un campo personalizado solo se puede añadir las descripciones traducidas en forma de un Map tipo Java, con clave de idioma y valor, al contrario que sucede con la descripción del artículo.
Para cada idioma que tenga el artículo, se crea un nuevo objeto JournalArticleDisplay, el cual proporciona la URL de la imagen completa, y el contenido del artículo en HTML para el idioma actual. La manera en la que se añadirá el contenido del artículo como descripción del evento es mediante un método que toma dos parámetros, el idioma y el contenido en una cadena de texto.
Queda pues, únicamente encontrar la manera de añadir la imagen al evento. Como los campos personalizados no admiten la subida de imágenes, no se podía hacer lo mismo

con la imagen que con el resumen, así que se decidió incrustar la imagen en la descripción del evento y obtenerla luego por JavaScript e incluirla en el componente **Event** así. En el campo de la descripción, en los detalles del evento, se oculta mediante una clase CSS. Esto se seguirá haciendo en los nuevos eventos que se creen, añadiendo la clase a la foto que quiera utilizarse de resumen.
Así pues, podemos añadir la descripción al evento con el método **setDescription(String description, Locale locale)**, siendo locale el idioma sobre el que se está iterando.
Para añadir el resumen no hay un método que lo añada por idiomas, como hemos comentado en un principio, así que tendremos que construir primero un mapa con todos los idiomas, y cuando se acabe el bucle de idiomas disponibles añadirlo entero con el siguiente método:

calendarBooking.getExpandoBridge().setAttribute("resumen",
calendarBookingResumen);

Por último, después de cada evento actualizado se deberán guardar sus cambios en la base de datos, con el método:

CalendarBookingLocalServiceUtil.updateCalendarBooking(calendarBooking);

Un comportamiento curioso de la base de datos es que guardaba los enlaces evento-contenido relacionado como pares, en uno estaba el evento primero y en otro era el artículo lo que estaba antes. La idea inicial fue borrar todos los enlaces que contenían el evento (que no el contenido relacionado), para así no tener que hacerlo dos veces, pero se decidió que no se debían borrar los eventos, en caso de que fueran necesarios en un futuro. La solución que tomamos al final para no borrar nada pero no realizar la misma acción dos veces fue simplemente ejecutar el código si el primer enlace era un evento del calendario. De esta manera, se puede reducir el tiempo de ejecución a la mitad.

5.11. Actualización de las preferencias del portlet

Al migrar un portlet todo debe estar igual que el antiguo para que el servidor no tenga problemas de referencias. Estas referencias no solo incluyen el título del portlet, sino que las configuraciones también deben tener el mismo nombe.

Realizando este análisis de configuraciones nos dimos cuenta de que había parámetros que se duplicaban para cada una de las vistas, que eran inútiles o que se quedaban obsoletas una vez se migrara la aplicación. Por ejemplo, la vista de tipo “Ambos” se estaba desechando, por lo que se tendría que modificar las preferencias que tuvieran ese valor.

Además, los parámetros de mostrar categorías o mostrar organización que eran independientes para cada una de las vistas, ya sea “Lista de eventos” o “Calendario”, y no era conveniente ya que en este portlet se ha pretendido unificar todas las configuraciones. La solución a la que llegamos fue mantener la configuración relevante al tipo de visualización actual del portlet. Esto quiere decir, que si se había especificado mostrar categorías en la “Lista de eventos”, pero no en “Calendario”, y la vista del portlet actual es “Calendario”, se debe mantener que no se muestren las categorías. Como cada parámetro tiene un nombre diferente, se ha creado un nuevo parámetro, por ejemplo mostrarCategorías, donde se añade el parámetro relevante para cada instancia.

El último paso de este proceso es actualizar la base de datos mediante los métodos proporcionados por la API de Liferay, como ha sido el caso en la ejecución anterior.

6. Validación

Dado que este proyecto se ha desarrollado bajo la tecnología de Liferay, lo principal para saber que todo ha salido bien es conseguir compilar y desplegar la aplicación en los servidores, primero en un entorno local y más adelante en un servidor propio de la GVA. Una vez desplegado, se probarán sus funcionalidades.

6.1. Compilación

Un portlet de Liferay acaba siendo, en esencia, un archivo **.jar** que deberemos incluir en la carpeta de módulos del servidor. Para obtener este archivo existen varias maneras, pero la más utilizada es mediante la herramienta Gradle, debido a su implementación en el IDE Liferay Developer Studio. En él, encontramos una pestaña dedicada exclusivamente a esta herramienta, y todas las tareas que se pueden ejecutar dentro de un portlet.

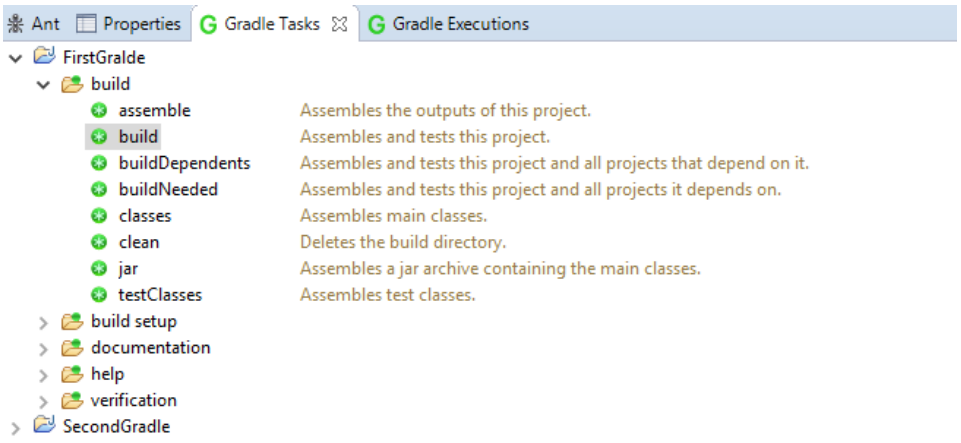

Figura 20: Cómo compilar un portlet con la herramienta Gradle.

Seleccionando la tarea *build* se generará un archivo jar, y se comprobará que no hay errores de sintaxis o dependencias.

Existen además otras herramientas útiles, como clean, que borra todo el directorio de compilación, en caso de que se necesite vaciar la caché, por ejemplo.

6.2. Despliegue

El despliegue de un portlet en los servidores de Liferay es muy sencillo. Basta con encontrar el archivo **.jar** compilado, generalmente dentro de la carpeta **build**, y copiarlo en la carpeta donde se aloje el servidor local que se llama **deploy.** Liferay permite el despliegue en "caliente", esto es, no es necesario reiniciar el servidor para actualizar un portlet preexistente en el mismo, lo cual ahorra mucho tiempo de programación.
Una vez desplegado ya se podrán hacer pruebas sobre la aplicación.

6.3. Compilación y despliegue en servidores de la GVA

En el caso de la Generalitat Valenciana, el despliegue no sigue el mismo proceso que en el entorno de desarrollo local.
Para posibles situaciones catastróficas, los desarrollos propios se deben desplegar desde la aplicación interan Jenkins de la GVA. Jenkins es un asistente de instalación que realiza todos los pasos por nosotros, haciendo en teoría más fácil todo este proceso.
Sin embargo, dado que el desarrollo del minicalendario ha sido el primer portlet que incluía dependencias de *node*, generaba problemas de compilación las primeras veces que se intentó instalar. Como desde nuestro lado no se podía hacer nada, se tuvo que solicitar ayuda al equipo de sistemas para que incluyeran el programa en una ubicación conocida en sus servidores, y desde nuestra aplicación apuntar a esa ubicación para que detectara la dependencia.

6.4. Pruebas

Las pruebas que se han realizado sobre la aplicación no tienen mucho misterio: A lo largo del desarrollo se ha ido comprobando que todo se ejecutaba correctamente, que la configuración de cada instancia coincidía con lo que estaba guardado en la base de datos, y que los eventos se mostraban correctamente por pantalla, con filtros de presentación y todas las posibles combinaciones.

Gracias a las pruebas realizadas fue como descubrimos la necesidad de migrar las configuraciones anteriores, y encontramos duplicidades en las mismas.

También tuvimos problemas a la hora de mostrar el detalle de los eventos: En un principio, se iba a visualizar mediante una ventana emergente, y no incrustado en el portlet. Al comentarlo con el cliente nos advirtió que tal visualización traía consigo problemas de accesibilidad, además de la imposibilidad de compartir los eventos mediante una URL personalizada. Así es como decidimos optar por la vista actual de los detalles.

Además, se realizaron presentaciones al cliente y diferentes usuarios editores para obtener *feedback* de ellos, y comprobar que el funcionamiento era coherente con lo que existía en un primer momento.

7. Conclusión

En esta memoria se ha detallado el proceso de diseño y desarrollo de una aplicación web desde cero para los portales de la Generalitat Valenciana bajo la tecnología de Liferay, estudiando e intentando mejorar el funcionamiento que tenía la versión anterior del portlet en los servidores actuales de la GVA. Se han implementado nuevas tecnologías, tanto para el equipo como a nivel personal, y se seguirá manteniendo hasta el despliegue en el entorno de producción, cuando todos los proyectos de migración se hayan completado.

Estudiando los objetivos que se requerían al principio del proyecto, podemos concluir que se han alcanzado en su totalidad. Recordamos que los objetivos eran los siguientes:

- Desarrollar una aplicación web que realice la visualización de los eventos de una forma similar al portlet ya existente en el servidor
- Mantener los datos ya existentes en la base de datos y visualizarlos sin problemas, realizando las modificaciones necesarias, tanto en la aplicación como en los objetos de la BBDD.
- Reducir el tiempo y la carga de ejecución de los portales en los que se incluya.
- Facilitar a los usuarios el uso de este portlet.
- Reducir al mínimo el trabajo que deben realizar los usuarios para utilizar esta nueva versión.
- Cumplir con los requisitos de accesibilidad y diseño *responsive* en la web.

El proceso de migración de los servidores no está, ni mucho menos, acabado. Aún falta mucho trabajo por hacer, aplicaciones, bases de datos, temas… Incluso ha salido una nueva versión de Liferay, la 7.3, durante el desarrollo del portlet minicalendario. Ésta, aunque menor, nos deja con la inseguridad de que la aplicación vaya a funcionar, debido a la falta de retrocompatibilidad por parte de los desarrolladores del gestor de portales. Será algo que comprobar en un futuro, antes de desplegarlo en el entorno de producción.

Un punto a criticar a nuestro cliente, sin embargo, es la falta de planificación. En mi opinión, en el momento en el que se actualizó Liferay a la versión 7.0, el inicio de Liferay DXP, se debería haber tomado en consideración iniciar este proyecto de migración. Sin embargo, al pasar tantos años, el trabajo ha ido aumentando, se han creado nuevos portales, mucho más contenido, ralentizando la fecha de fin del proyecto.

7.1. Relación del proyecto con los estudios cursados

Durante el desarrollo del proyecto se ha podido aplicar mucho de lo que se estudió en la carrera, además de lo aprendido gracias a participar en un equipo profesional.

Gracias a las asignaturas de Gestión de proyectos e Ingeniería del Software hemos podido diseñar un plan de desarrollo de la aplicación, planificando el trabajo a realizar, controlar dicho trabajo y cumplir con las entregas del portlet.

Con las asignaturas de Bases de datos, afianzamos los conocimientos necesarios para trabajar con BBDD y estructuras de todos los lenguajes con los que hemos tratado, además de técnicas de recuperación de la información necesaria para el funcionamiento de la aplicación.

La asignatura Interfaces persona computador nos ayudó a entender cómo diseñar una buena interfaz para nuestra aplicación, cómo buscar las reglas de accesibilidad y aplicarlas en nuestro proyecto. En esencia, el resultado de esta aplicación depende exclusivamente de su buena interfaz, y se ha intentado conseguir la mejor versión de la misma.

Hablando de las competencias transversales conseguidas en esta carrera, podríamos destacar el análisis y resolución de problemas, que ha sido el pilar central en este proyecto. También la de comunicación efectiva, a la hora de reunirnos con el cliente y presentar los avances de la aplicación cuando ha sido necesario.

En definitiva, soy completamente consciente de la importancia que ha tenido la carrera para el éxito en mi primer contacto con el mundo laboral real, y agradezco mucho las decisiones que me han llevado a estar aquí.

8. Trabajos futuros

Pese a que el desarrollo del portlet minicalendario ha finalizado, no ha dejado de ser una preocupación para el equipo. El gestor de portales Liferay ya se ha actualizado a una versión 7.3 y, pese a que el cambio no es tan abismal como la diferencia entre la versión 6.1 a la ya estudiada 7.2, es necesario comprobar que ninguna de las funcionalidades del portlet minicalendario ha quedado obsoleta.

Además, nuestra aplicación aún no ha visto la luz del día en los entornos de producción. Queda mucho trabajo de migración antes de actualizar los servidores de la GVA. En el momento en el que se deba desplegar, deberemos analizar las opiniones de los usuarios, tanto visitantes como editores, para comprobar que todo ha salido como esperaba, o modificar aquellas características que lo requieran.

9. Referencias bibliográficas

[1]. Boletín Oficial del Estado (2014, 10 julio) *Anuncio de la Conselleria de Hacienda y Administración Pública por el que se publica la formalización del contrato administrativo de Servicio en Informática: Proyecto Desig-2013 (Desarrollo de los sistemas de información de la Generalitat).* https://www.boe.es/diario_boe/txt.php?id=BOE-B-2014-26738

[2]. Liferay (s.f.) *Liferay DXP* https://www.liferay.com/es/products/dxp/

[3]. Liferay (2019) *What Hasn't Changed and What Has* https://help.liferay.com/hc/es/articles/360018156891-What-Hasn-t-Changed-and-What-Has

[4]. Liferay (2018) *Liferay Events List* https://web.liferay.com/es/marketplace/-/mp/application/83511153

[5]. Liferay (2021) *Calendar Events Enhancement* https://web.liferay.com/es/marketplace/-/mp/application/89449030

[6]. NPM (2021) *React-Calendar* https://www.npmjs.com/package/react-calendar

[7]. Liferay (2019) *What is Service Builder?* https://help.liferay.com/hc/es/articles/360018160851-What-is-Service-Builder-

[8]. Liferay (2020) *npm React Portlet Template* https://help.liferay.com/hc/es/articles/360017902132-npm-React-Portlet-Templatehttps://help.liferay.com/hc/es/articles/360017902132-npm-React-Portlet-Template

[9]. Liferay (2020) *Finding and Invoking Liferay Services* https://help.liferay.com/hc/es/articles/360018160991-Finding-and-Invoking-Liferay-Services

[10]. ReactJS (s.f.) *Estado del componente* https://es.reactjs.org/docs/faq-state.html

10. Glosario de términos

- **JSON**: Acrónimo de JavaScript Object Notation, es un formato de texto sencillo para el intercambio de datos.
- **Portlet**: Aplicación web desarrollada bajo el entorno de Liferay. Es un componente modular que se puede añadir a los portales donde esté instalada.
- **BBDD**: Siglas para base de datos.
- **XML**: eXtensible Markup Language, es un metalenguaje que permite definir lenguaje de marcas, utilizado para almacenar datos de forma legible.
- **GVA**: Siglas para Generalitat Valenciana.
- **Backend**: Parte del desarrollo web que se encarga de toda la lógica para que una página web, en este caso, funcione.
- **Frontend**: Parte del desarrollo web que consiste en la visualización de los datos obtenidos mediante el *backend*.
- **Feedback**: Acción de ofrecer información a una persona sobre un resultado. En este caso, el cliente y los usuarios editores han dado *feedback* sobre el desarrollo del minicalendario.

11. Apéndices

11.1. Ejemplo de un objeto JSON con eventos de múltiples calendarios


```
{
  "El montgó": {
    "20/02/2021": [
      {
        "calendarBookingId":12345,
        "calendarId":4132,
        "calendarName":"El montgó",
        "title":"Paseo por la Cova de l'Aigua.",
        "description":"Lorem ipsum dolor sit amet",
        "startDate":"20/02/2021",
        "duration":11123300,
        "resumen":"Lorem ipsum"
      },
      {
        "calendarBookingId":14458,
        "calendarId":4132,
        "calendarName":"El montgó",
        "title":"Paseo por la Cova del Gamell.",
        "description":"Lorem ipsum dolor sit amet",
        "startDate":"20/02/2021",
        "duration":11123300,
        "resumen":"Lorem ipsum"
      }
    ],
    "25/02/2021": [
      {
        "calendarBookingId":258964,
        "calendarId":4132,
        "calendarName":"El montgó",
        "title":"Charla de concienciación de los incendios forestales",
        "description":"Lorem ipsum dolor sit amet",
        "startDate":"25/02/2021",
        "duration":3123300,
        "resumen":"Lorem ipsum"
      }
    ]
  },
  "El Túria": {
    "19/03/2021": [
      {
        "calendarBookingId":12345,
        "calendarId":4132,
        "calendarName":"El Túria",
        "title":"Juegos libres en el Parque Gulliver.",
        "description":"Lorem ipsum dolor sit amet",
        "startDate":"19/03/2021",
        "duration":50123300,
        "resumen":"Lorem ipsum"
      }
    ]
  }
}
```


Figura 21: Ejemplo de un objeto JSON con eventos de múltiples calendarios.