Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

# Diseño e implementación de una aplicación para el cálculo de la fecha máxima de entrega de paquetes

Trabajo Fin de Grado

**Grado en Ingeniería Informática**

**Autor**: Pablo Cerro Montoya

**Tutor**: Ana Pont Sanjuan

Curso 2020-2021

# Resumen

El contexto tecnológico al que se dirige este proyecto, parte de un ámbito económico en el cual las empresas optan por arquitecturas orientadas a servicios y priman los e-Business ante las arquitecturas monolíticas y el comercio físico.

En un mundo donde la inmediatez está a la orden del día y la información es primordial, las empresas optan por desarrollar nuevos servicios, en vez de utilizar los ya implementados por otras empresas. En muchos casos se gastan recursos en nuevas infraestructuras para servicios con funcionalidades ya implementadas por otros. Por ello, se contempla la idea de englobar uno de los servicios más empleados durante los últimos años en un único servicio capaz de integrarse en cualquier tecnología. Dicho servicio se trata de la generación de fechas de entrega máxima.

Durante el transcurso del siguiente documento se mostrará el diseño paso a paso de un servicio web capaz de generar y comunicar fechas de entrega máximas para empresas o particulares. Además, dicho servicio podrá ser integrado con cualquier tecnología para su uso, hecho que facilita el empleo del servicio en empresas con web propias, ya que será sencillo añadir en ellas la información generada.

**Palabras clave:** servicio web, compras on-line, fecha de entrega máxima

# Abstract

In the technological context to which this project is directed, it starts from an economic environment in which companies opt for service-oriented architectures and e-Business prevail over monolithic architectures and physical commerce.

In a world where immediacy is essential and information is paramount, companies develop new services on their own. In many cases infrastructure is used for services with the same functionality. For this reason, the idea of encompassing one of these services, the generation of maximum delivery dates, in a single service capable of being integrated into any technology is contemplated.

During the following document, the step-by-step design of a web service capable of generating and communicating maximum delivery dates for companies or individuals will be shown. In addition, said service may be integrated with any technology for its use, a fact that facilitates the use of the service in companies with their own websites, since it will be easy to add the information generated to them.

**Keywords:** web service, on-line purchases, maxium delivery date.

# Tabla de contenidos

# Tabla de Ilustraciones

# 1. Introducción

Durante mucho tiempo los negocios han ido evolucionando hasta el punto donde los comercios ya no solo se basan en la venta física de sus productos, sino que también ofertan sus productos en la red. Este contexto económico genera la necesidad de crear tiendas virtuales para ofrecer sus mercancías con el fin de hacer frente a la competencia.

En el ámbito de la oferta de productos online juega un gran papel la información. ¿Qué calidad tiene el producto?, ¿Me lo traen a casa o debo acercarme a la tienda más cercana a recogerlo?, ¿Cuánto tiempo debo esperar hasta poder recibir mi compra?

Hay mucha de esta información que el propio ofertante puede exponer en su web para atraer a sus clientes, pero hay otra de la cual carece como es el caso de la fecha de entrega del paquete en el momento que el usuario se interesa por el producto ofertado.

La fecha de entrega máxima de paquetes es un factor decisivo en la toma de decisiones del cliente ya que mediante dicha información se establece, a través de la web, un compromiso entre la empresa y el cliente. Aunque parezca algo banal, esta información provee al servicio de una cierta profesionalidad que genera en el cliente seguridad a la hora de realizar su compra.

La información acerca de la fecha de entrega de un producto muchas veces no depende del comercio, sino de la empresa de transporte con la que se trabaje. Esto implica que cada comercio tendrá unos plazos de entrega diferentes debido a que esto varía en función de la empresa de transporte con la que se trabaje y la tarifa contratada.

En respuesta a esta situación el autor del trabajo propone centralizar todos los plazos de dichas empresas de transporte en una fuente de información, capaz de atender a cualquier dispositivo conectado a internet, que a través de una serie de parámetros es capaz de generar la fecha de entrega máxima de un paquete”.

## 1.1 Objetivos

El objetivo de este trabajo es generar un servicio que integre toda la información de empresas de transporte y provea al usuario información sobre la fecha de entrega de mercancías adecuándose al horario y los festivos de la tienda que oferta el producto. Además, dicha información debe resultar útil a la hora de exponer sus productos en la web, sea fácil de comunicar al posible cliente, proporcione una sencilla interfaz de mantenimiento por parte de las tiendas y las empresas transportistas y, por último, ofrezca una buena fiabilidad.

## 1.2 Estructura

El siguiente trabajo académico presenta el resultado de la investigación del autor y el proceso de desarrollo de un producto software. Dicha presentación va a estar dividida en varios apartados, o capítulos, en los que detallaré mis reflexiones y conclusiones.

En el primer apartado, la introducción, se expondrá la idea del autor desarrollada durante el transcurso de la memoria. Además, se mostrará a grosso modo el fin último de este desarrollo o, dicho de otra manera, los objetivos que toma como punto de partida el proyecto.

En el segundo apartado, el estado de la cuestión, se recopilará toda la información referente el estado actual del mercado para obtener una idea del impacto o acogida resultante de la realización del proyecto. Después, se analizará y se llevará a cabo una propuesta capaz de cumplir los objetivos anteriormente definidos.

A partir de este punto, se realizará una reflexión en profundidad sobre el problema que se va a abarcar para así en el tercer apartado, el análisis del problema, poder definir el alcance la solución y poder generar una planificación acorde.

En el cuarto apartado, el diseño de la solución, se muestra todo los detalles y características que abarcará el producto resultante de este proyecto como la arquitectura del sistema, la interfaz de usuario y las tecnologías con las que se implementará el diseño anteriormente creado.

Una vez definida la parte teórica del proyecto se pasará al quinto apartado, el desarrollo de la solución, donde se mostrarán de forma expositiva los pasos que se han realizado para la creación del producto.

En el sexto apartado, la implantación, se detallará la cadena de procesos que debe realizase para desplegar el producto desarrollado en algún sistema que sea accesible desde el exterior por el público interesado.

Acabada la parte práctica del proyecto se continuará con el séptimo apartado, las pruebas, dónde se redactarán los resultados de los diferentes ensayos a los que se someterá el producto final. Y finalizaremos con las conclusiones y futuros trabajos, se verá si se han cumplido los objetivos planteados anteriormente y se detallarán los futuros trabajos que no se han tratado durante este proyecto.

## 1.3 Convenciones

En esta sección se muestran las principales convenciones adoptadas a lo largo de la presente memoria:

- Los nombres de los ficheros importantes para la descripción o configuración de las tecnologías estarán escritos en letra `Courier New` para resaltar. Como por ejemplo el `standalone.xml`.

- Utilizaremos el tipo de letra Abadi Extra Light para destacar los nombres que hacen referencia a imágenes en el proyecto como el caso de CFME_FRONT en la imagen de la arquitectura del sistema.

- Para citar partes de código, comandos o rutas relevantes para el desarrollo del proyecto se empleará el entrecomillado simple como es el caso de ‘docker pull wnameless/oracle-xe-11g-r2’.

- Los nombres de los formatos de los archivos se mostrarán en mayúscula, asumiendo que son acrónimos del inglés. Ejemplos: XML, EAR, JAR, etc.

- Las palabras extranjeras se mostrarán en letra cursiva.

# 2. Estado de la cuestión

Durante el transcurso de este apartado se dará a conocer el contexto general al que se dirige el proyecto. Para ello, se expondrá información acerca de aplicaciones similares y el estado actual la economía.

Comenzando con el campo más amplio la economía, nos encontramos ante lo que se podría denominar "La economía digital", también conocida como "Economía de Internet" o "Economía Web", término que utilizó el autor Trapscott en su libro más vendido en 1995, *La Economía Digital: Promesa y peligro en la Era de la Inteligencia en redes* [1].

La economía digital está formada por la infraestructura de telecomunicaciones, las industrias TIC (*software, hardware* y servicios TIC) y la red de actividades económicas y sociales facilitadas por Internet, la computación en la nube y las redes móviles, las sociales y de sensores remotos [2].

En dicha economía, de acuerdo con Thomas L. Mensenbourg, encontramos tres elementos principales [3]:

- Los negocios electrónicos(e-Business). Esta forma de negocio facilita las relaciones entre empresas permitiendo así compartir servicios entre ambas a través de interconexiones.
- Las empresas electrónicas(e-Enterprises). Estas empresas substituyen los métodos tradicionales por los electrónicos y basados en la web.
- El comercio electrónico (e-Commerce). En este tipo de comercio los bienes y servicios se sustituyen por otros electrónicos que otras empresas pueden integrar en ellas

Por otro lado, hay consecuencias que derivan de la economía digital que debemos tener en cuenta como son: la composición de esta forma de economía, que facilita a las empresas la subcontratación de otras para evitar crear servicios que realizan las mismas funciones eludiendo así sobrecostes y, además, un mercado donde la inmediatez se ha convertido en un requisito indispensable para los clientes.

En este contexto donde las empresas migran sus métodos tradicionales a la Web, encontramos un componente muy importante, los Servicios Web. Dichos servicios son, según Alex Rodríguez [5] "REST define un conjunto de principios arquitectónicos por los que se pueden diseñar servicios Web que

se centran en los recursos de un sistema, lo que incluye la forma en que los estados de los recursos se dirigen y transfieren a través de HTTP por un amplio rango de clientes que están escritos en diferentes lenguajes". Además, esta definición puede ser ampliada con la proporcionada por SUN Microsystems [6]: "un servicio web es una aplicación que existe en un entorno distribuido como Internet. Dicho servicio acepta una petición, realiza una función basada en la petición, y devuelve una respuesta".

Los servicios que acabamos de definir son utilizados por muchas grandes empresas como por ejemplo Amazon ya que aportan muchos beneficios a la hora de organizar empresas. Uno de estos beneficios es la posibilidad de generar relaciones B2B o *business to business*, traducido como negocios con negocios, que consisten en realizar servicios entre empresas para aumentar las ventas de productos o servicios como se observa en la Ilustración 1.

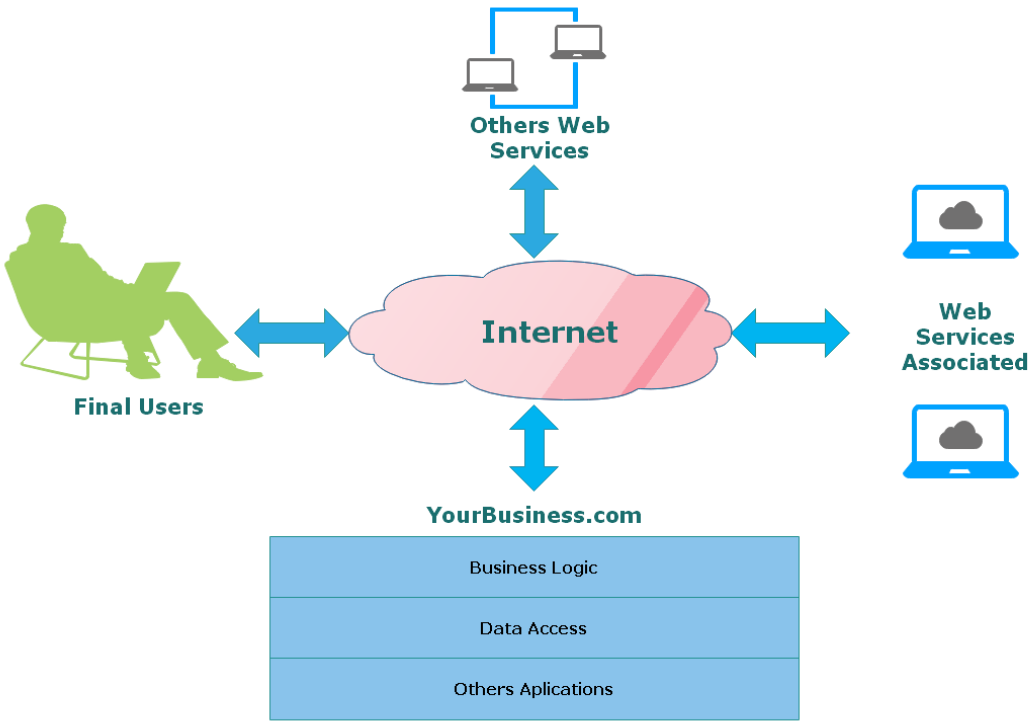



Ilustración 1 Modelo B2B

Como ya hemos visto anteriormente las empresas desarrollan sus servicios web o utilizan los servicios de otras empresas. En concreto, el servicio de cálculo de fechas de entrega es utilizado por muchas empresas para dar información a sus clientes, pero cada empresa desarrolla el suyo propio. En cambio, existe en la web una aplicación que engloba todos estos

servicios en uno, PackLink[1]. En esta aplicación se puede consultar con que empresa te saldrá más económico gestionar tu envío y te muestra el tiempo que tardará en ser entregado.

La aplicación de PackLink tiene una funcionalidad similar a la que se va a desarrollar en este proyecto el cual consta de dos partes: la primera para usuarios que funciona mediante introducción de parámetros en la web generará una respuesta acorde a los parámetros de la solicitud. La segunda, la parte para empresas, PackLink Pro[2], es una plataforma que está integrada con Amazon, Ebay, Correos y muchas otras para permitir de forma eficaz gestionar todos los envíos de una empresa a sus clientes.

PackLink es un gran ejemplo de modelo de aplicación actual y que incluye la funcionalidad del cálculo de fecha de entrega de paquetes, pero como fin último, es decir, no puede ser integrada con otras plataformas. Esta funcionalidad se asimila mucho a la que se va a desarrollar en el siguiente proyecto, ofreciéndola de manera desacoplada y personalizada, tanto para los clientes como para las empresas de transporte.

## 2.1 Propuesta

La propuesta que se va a desarrollar es un servicio web que centraliza toda la información acerca de las diferentes empresas de transporte de paquetes que trabajen a nivel nacional y, además, sea capaz de generar fechas de entrega máxima de paquetes con buena fiabilidad.

En este servicio las empresas tendrán la opción dar de alta sus oficinas facilitando su localización a través de un código postal y sus horarios de apertura. Si escogen esta opción podrán usar el servicio web como un generador de fechas de entrega para su negocio, pudiéndolas comunicar a sus clientes mediante su web.

En el caso de no realizar esta opción, el servicio web será un generador de fechas de entrega similar a PackLink es decir, un servicio que devolverá una fecha de entrega cuando se le facilite en un código postal de origen y uno de destino.

Esta propuesta, a diferencia de la aplicación anteriormente presentada como ejemplo, no permitirá gestionar los envíos solo se enfoca a la generación e información de la fecha de entrega máxima de paquetes.

[1] https://www.packlink.es/
[2] https://pro.packlink.es/

# 3. Análisis del problema

Durante el transcurso del presente apartado se entrará en detalle sobre las oportunidades de negocio y necesidades que este proyecto abarca. A lo largo de este apartado focalizaremos y mostraremos hacia dónde va dirigida nuestra propuesta.

Como ya se ha comentado anteriormente cualquier empresa, que oferte productos físicos mediante la web, debe poder informar a los clientes de las fechas de entrega cuando se le realice un pedido. Para ello, el sistema generador y comunicador de dichas fechas debe permanecer siempre activo y ser capaz de atender al volumen de peticiones que reciba, es decir, debe ser un sistema robusto y escalable.

Por otro lado, la entrega de paquetes no tardará lo mismo si la petición se hace a una empresa que tiene un horario de trabajo de 24h que a una empresa que trabaje 8 horas. Tampoco tardará lo mismo una empresa en Valencia cuando son Fallas, que una en Madrid, ya que en una localidad es festivo y en otra no. La fecha de entrega de paquetes es una información que varía entre empresas, así que el sistema deberá adecuarse a sus características.

Además, un calculador de fechas debe de tener en cuenta que las empresas de transporte ofrecen servicios que pueden variar con el tiempo. Así, el sistema debe de poder ser modificado para satisfacer estas necesidades de forma sencilla e inmediata.

Por último y más importante, la fecha de entrega calculada tiene que ser lo más realista posible. Si la fecha calculada es muy lejana, la opción de comprar a través de esa empresa puede ser descartada por ser demasiado tardía. En cambio, si la fecha calculada es muy cercana, dicha empresa puede no llegar a cumplirla, rompiendo así el contrato establecido entre ambas partes.

## 3.1 Identificación y análisis de posibles soluciones

El problema ya se ha sido definido en el capítulo anterior, así que en el siguiente se buscarán premisas que permitan dar una solución a dicho problema. Se comenzará dando solución a los problemas relacionados con la arquitectura y después a los relacionados con la lógica.

La solución que se busca para la arquitectura es un servidor optimizado para servicios web, que soporta una gran cantidad de peticiones sin colapsar y sea capaz de atenderlas todas en un espacio de tiempo tolerable.

Para definir este tiempo tolerable recurriremos a la definición explicada por el autor Jakob Nielsen en su libro *Usability Engineering*[7] donde define tres intervalos de tiempo que hay que tener en cuenta a la hora de optimizar una web. El primer límite es 0.1 segundos que es donde el usuario siente que el sistema reacciona instantáneamente a sus acciones. El segundo limite es de 1 segundo hasta aquí el usuario pierde la sensación de estar manejando directamente el sistema y comienza a sentir una sensación de retraso en la respuesta, pero en este punto no es necesario emplear técnicas para que el usuario se mantenga interesado. Por último, el tercer límite son los 10 segundos donde es necesario el empleo de técnicas para mantener el interés del usuario como son las barras de progreso y herramientas similares. Además, Google define en su página de desarrolladores [3] un tiempo óptimo de respuesta de doscientos milisegundos y uno tolerable de hasta quinientos.

Por ello, se entiende como tiempo de respuesta razonable para el servició de generación de fechas de entrega el intervalo entre 0 y 0.5 segundos desde la acción del usuario. En cambio, el intervalo de respuesta para la aplicación web puede variar de 0 a 10 segundos ya que esta sí que permite informar del estado del sistema al usuario haciendo así la espera más amena evitando así que el usuario pierda el interés.

Siguiendo estas especificaciones hemos encontrado el servidor de aplicaciones java llamado Wildfly[4] (antiguamente conocido como Jboss). Este tipo de servidor tiene una alta disponibilidad, es decir, funciona con múltiples instancias que trabajan conjuntas para ser más resistente a fluctuaciones, cargas y fallos del servidor. El concepto de alta disponibilidad también hace referencia a la escalabilidad, balanceo de

[3] https://developers.google.com/speed/docs/insights/Server
[4] https://www.wildfly.org/

carga y tolerancia. Esta característica hace del servidor Jboss un servidor de aplicaciones idóneo para albergar en él nuestro servicio web.

Una vez seleccionado el servidor, ahora se buscarán soluciones para que el cálculo de la fecha contenga las características expuestas en el apartado anterior.

La solución aportada se fundamenta en un algoritmo basado en dos puntos, un origen y un destino. Dichos puntos estarán separados por una distancia con un valor determinado que variará dependiendo de la tarifa de la empresa de transportes con la que se trabaje. Por otro lado, se tendrá en cuenta los festivos que transcurran desde la admisión del paquete hasta la entrega de este. Además, la admisión y la entrega deberán transcurrir durante el horario activo de las empresas que soliciten la generación de la fecha de entrega máxima del paquete.

Los datos que se necesitarán serán: los festivos locales, provinciales, autonómicos y nacionales relacionados con las provincias que afecten, todas las provincias de España relacionadas con su número de provincia y todos los códigos postales de España relacionados con la provincia a la que pertenecen. Estos datos se pueden localizar en fuentes de datos abiertos como la plataforma OpenDataSoft[5].

Para relacionar los códigos postales con sus provincias se pueden tomar los dos primeros dígitos de los cinco que componen el código postal español. Estos dígitos hacen referencia a la provincia, asignados siguiendo un orden alfabético, del 01 al 50, y a las ciudades de Ceuta y Melilla que, al estar fuera de la división provincial, se les asignaron el 51 y el 52 respectivamente.

Con todos estos datos ya se puede generar una solución acorde con las especificaciones anteriormente mencionadas y se pasará a la solución del siguiente reto, la variabilidad de los servicios de transporte o de los horarios de las empresas.

Para dar solución a dicho inconveniente debemos buscar un lugar donde tengan acceso tanto las empresas que ofertan productos como las que ofrecen el transporte. La web es un sitio que cumple con ese requisito, así que la solución es una aplicación web que sea capaz de modificar los datos de las empresas y personalizar el servicio a las empresas.

[5] https://data.opendatasoft.com/

## 3.2 Solución propuesta

Para dar solución a los problemas expuestos con anterioridad, se propone implementar por razones de disponibilidad un sistema compuesto por: dos servidores, uno que albergará la aplicación web y otro que contendrá el servicio de generación de fechas. Un servidor hará la función de backend recibiendo las solicitudes de generación de fechas de entrega máxima y realizando el mantenimiento informado a través del otro servidor. El segundo servidor hará la función de front-end exponiendo una interfaz amigable al usuario a través de la cual se podrán modificar y crear tarifas y dar de alta empresas que requieran del servicio de generación de fechas.

Ambos servidores se comunicarán entre ellos para la modificación de la información de las empresas que empleen el servicio mediante el protocolo HTTP. Dicho protocolo permite la comunicación entre dispositivos que no comparten tecnologías, es decir, cualquier tecnología es admitida por este protocolo. Esto permitirá al servicio integrarse con cualquier plataforma desde la más antigua hasta la más novedosa.

También será necesaria una base de datos que almacene todos los datos mencionados en el apartado 3.1 que contenga una conexión al servidor backend. Por último, el servido front-end dispondrá de una aplicación web implementada con tecnología jsx para la modificación, inserción o eliminación de los datos referentes a las empresas. Se ha elegido esta tecnología para per ampliar los conocimientos del autor ya que se basa en una mezcla de JavaScript y HTML orientada a componentes de fácil mantenimiento y uso muy intuitivo.

## 3.3 Plan de trabajo

En el este apartado se explica las fases en las cuales se ha planificado el desarrollo del proyecto, es decir, mediante la Ilustración 2 se mostrarán las diferentes etapas distribuidas a lo largo del periodo definido para el desarrollo del proyecto presentado en el presente documento.

La planificación realizada para el proyecto abarca tres meses y contiene cuatro tareas principales divididas en varias subtareas. Además, está planificada para que el trabajo se reparta de lunes a viernes con horario de “09:00:00 a 14:00:00”, es decir, en jornadas de cinco horas.

Como se puede observar en Ilustración 2, la primera tarea que se trata en la planificación es la investigación, fase durante la cual se buscará información acerca de las tecnologías actuales, el contexto al que dirigimos nuestro proyecto y datos clave para la creación del algoritmo. Esta primera fase tendrá una duración aproximada de diez días porque partimos con una idea clara de lo que se busca.

Una vez realizada la tarea de investigación, según la planificación la siguiente fase es la de diseño en la cual, durante diez días, aplicamos toda la información recopilada en la tarea anterior y generamos el diseño de un sistema capaz ofrecer la funcionalidad esperada. Durante el diseño separamos los distintos componentes que conforman nuestro sistema como son la base de datos, la arquitectura del servicio y la interfaz web.

La tercera fase, la implementación consiste en la creación de un sistema siguiendo las pautas especificadas en el diseño. Esta etapa diferencia, al igual que la de diseño, los diferentes componentes a implementar como son la instalación del entorno, la creación del proyecto y la implementación de la lógica.

En la subtarea instalación del entorno se incluye la preparación del puesto de trabajo, es decir, la creación de un entorno que se disponga de todas las herramientas necesarias para el desarrollo del proyecto. Dentro de estas herramientas se encuentran las tecnologías para la creación y gestión de la base de datos, las de desarrollo, las de integración y las necesarias para realizar las pruebas.

La subtarea de creación del proyecto hace referencia a la fase de creación del programa mediante las herramientas instaladas en la fase anterior. Durante esta fase se integran todas las tecnologías en nuestro entorno de desarrollo y se genera la arquitectura del proyecto siguiendo las pautas definidas en el diseño.

Y para terminar la fase de implementación se realiza la subtarea de implementación de la lógica, en la que se programa el algoritmo diseñado y se genera un ejecutable o un programa funcional con el que se puede comenzar a depurar para lograr el producto final deseado.

En este punto, la fase de implementación ha terminado y se comienza con la fase de pruebas donde a través de la técnica de *debug* y el lanzamiento de peticiones en un entorno controlado se puede depurar el programa hasta lograr el funcionamiento deseado.

Para concluir con la planificación, cabe destacar una última tarea que comienza cuando termina la fase de investigación, la fase de documentación. Esta tarea consiste en dejar por escrito los resultados de las tareas llevadas a cabo. Además, esta se realiza en paralelo con todas las restantes para conseguir un resultado más preciso, ya que, si se documentan los resultados o los procesos en el momento, se logra una descripción con mayor detalle y precisión.

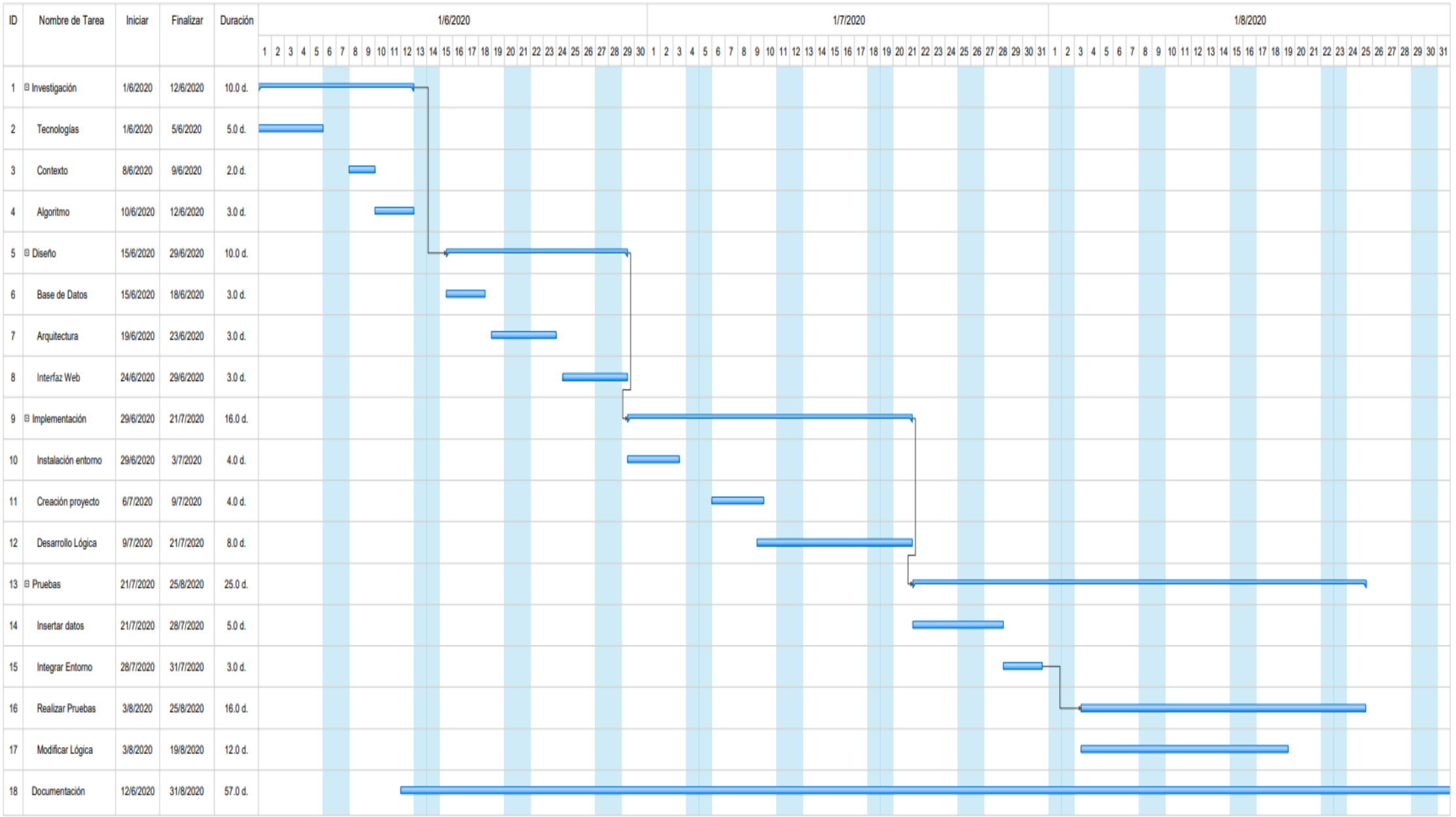



*Ilustración 2 Diagrama de Gantt con la distribución de las tareas*

# 4. Diseño de la solución

En el presente apartado se mostrarán los diseños de las diferentes partes que componen el sistema. Estas partes son: los servicios web diseñados mediante un esquema de Arquitectura Lógica, la base de datos diseñada mediante un esquema relacional, las interfaces web diseñadas mediante la técnica del *mockup* y la arquitectura software diseñadas siguiendo el modelo de arquitectura a tres niveles.

## 4.1 Arquitectura del sistema

El sistema desarrollado consta de un servicio web, implementado en JAVA que se ubica en un servidor JBoss el cual atenderá las solicitudes y comunicará las respuestas a los clientes.

El programa o software está implementado en un modelo a tres capas diferenciando así la capa de más externa, la interfaz, de la capa media, la lógica de negocio, y la capa final, la capa de datos. Cada capa tendrá su utilidad y dicha diferenciación ayudará a la organización del código y en caso de querer ampliar la funcionalidad del programa facilitará su futura escalabilidad.

La primera de las capas que hemos comentado, la interfaz, es una pantalla donde se exponen los diferentes servicios que puede ejecutar nuestro programa. Dicha pantalla alberga la documentación de los parámetros de la petición y de la respuesta para facilitar a futuros clientes su uso y a futuros programadores su mantenimiento.

La segunda capa, la lógica de negocio, es donde se sitúan todas las funciones o servicios. Además, alberga todo lo relativo al tratamiento de las peticiones y es la encargada de generar una respuesta acorde.

Y por último la tercera capa, la capa de datos, que es la encargada de recoger o modificar los datos de la base de datos y comunicar los resultados a la capa de negocio, es decir, es la capa donde se tratan todas las interacciones con la base de datos.

Para almacenar los datos de este servicio se ha elegido una base de datos relacional. En este caso se ha optado por una base de datos de Oracle donde se ha diseñado un modelo de datos para el correcto funcionamiento de la aplicación.

Por otro lado, se encuentra una interfaz web para el mantenimiento de los datos del servicio, es decir, una aplicación web a través de la cual las empresas pueden modificar o dar de alta sus datos.

## 4.2 Diseño detallado

En este apartado detallaremos todo lo relativo al diseño del servicio web. Comenzaremos con una explicación sobre el diseño de la arquitectura lógica y con los diseños de la interfaz web, continuaremos con el diseño del modelo de datos que se ha implementado, seguiremos con el diseño de las peticiones y respuestas del servicio y terminaremos mostrando el algoritmo implementado para la generación de la fecha de entrega máxima del paquete.

### 4.2.1 Arquitectura lógica

Para la descripción de la arquitectura se emplearán los nombres asociados a la Ilustración 3.

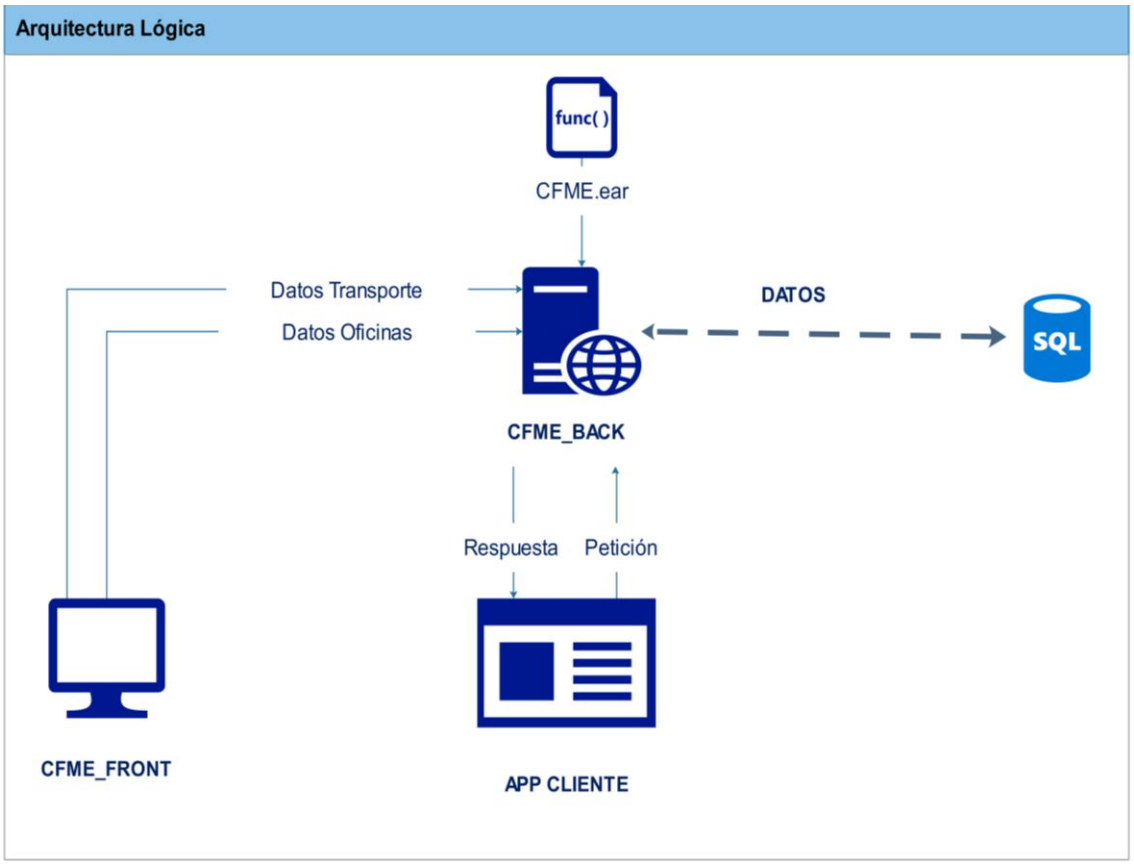



*Ilustración 3 Arquitectura Lógica*

La arquitectura lógica tiene dos servidores CFME (Calculador de Fecha Máxima de Entrega), un Apache CFME_FRONT, que contendrá la aplicación web, y un JBoss CFME_BACK, que contendrá el servicio web. Este segundo servidor implementará toda la funcionalidad del servicio, recibiendo peticiones y enviando respuestas a los distintos servidores que contacten con él y persistiendo la información comunicada a través del CFME_FRONT, es decir, hará la funcionalidad de *back-end*.

El servidor CFME_FROTN es un servidor Apache en el cual se desplegará una aplicación web implementada con REACT[6] utilizando componentes de las librerías de Ant Design[7]. Además, a través de ella se podrán consultar modificar y eliminar datos, es decir, hará la función de *front-end*. Dicha aplicación constará de cinco pantallas, tres para el mantenimiento y otras dos para el alta.

### 4.2.2 Modelo Interfaz Usuario

A continuación, se mostrarán los diseños de las pantallas que compondrán la interfaz web que será alojada en el servidor CFME_FROTN.

Comenzamos con el diseño mostrado en la ilustración 4, el diseño de la ventana de mantenimiento de rutas. En esta pantalla las Empresas de transporte podrán editar los datos que hayan dado de alta con anterioridad. El filtro busca de entre todas las empresas (campo obligatorio), tarifas y provincias para facilitar así la localización del registro a consultar. Una vez rellenados los campos del filtro si el usuario hace clic en buscar el sistema le muestra una tabla con todos los registros que coincidan con los campos preseleccionados. En caso de hacer clic sobre limpiar, el sistema vaciará los campos del filtro y hará desaparecer la tabla, en caso de que se estuviera mostrando.

Una vez realizada la búsqueda y dado con el registro deseado, el usuario puede realizar dos acciones, eliminar el registro o editarlo. En ambos casos el sistema mostrará una alerta de alteración de datos que el usuario deberá confirmar. Si es confirmada el sistema mostrará el resultado de la operación, en caso contrario se cerrará la alerta y el sistema no continuará con la acción.

[6] https://es.reactjs.org/
[7] https://ant.design/components/overview/

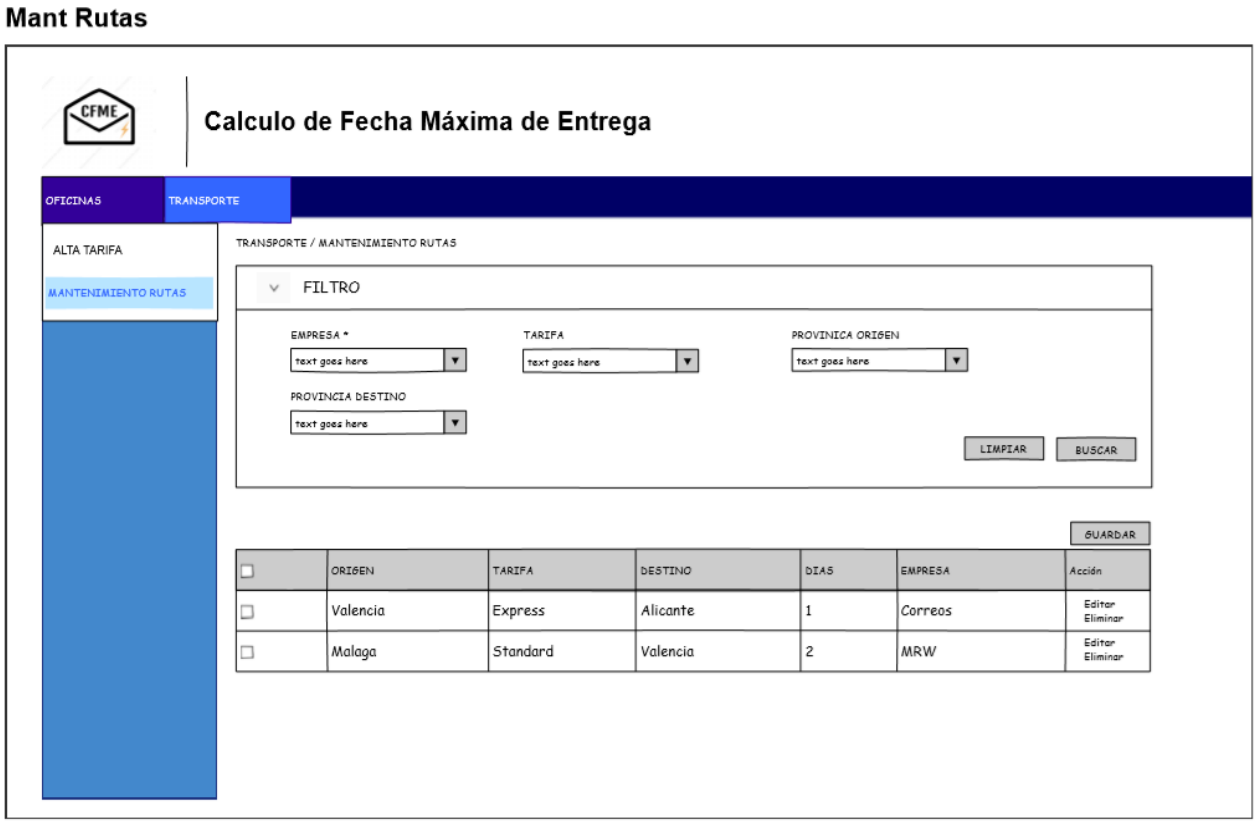



*Ilustración 4 Diseño Web: Mantenimiento Rutas*

La interfaz de Alta Transporte se utilizará para la creación de tarifas nuevas o para agregar registros a tarifas ya dadas de alta. Además, si se trata de una nueva empresa de transporte que quiere dar de alta su primera tarifa se habilita un *checkbox* para satisfacer su necesidad.

La interfaz ofrece dos caminos para la adicción de registros para las nuevas tarifas. Uno por importación de un Excel con los mismos campos que la tabla mostrada en la interfaz y otro por inserción de datos uno a uno vía web. Esto ayuda a nuevas tarifas añadir gran cantidad de registros con facilidad y para agregar algún registro a tarifas ya existentes poder hacerlo rápidamente.

Para añadir uno a uno el usuario deberá rellenar los campos obligatorios y hacer clic sobre el botón Añadir Registro y el sistema lo añadirá a la tabla. Para importar los registros desde un Excel deberá seleccionar el fichero a subir haciendo clic en el botón Importar Datos y el sistema comunicará el número de registros que se han detectado del fichero.

Una vez se han introducido todos los datos deseados el usuario podrá hacer clic en Aceptar y el sistema mostrará la alerta de alteración de datos y comunicará el resultado de su operación. En cambio, si hace clic en Limpiar el sistema eliminará todos los datos introducidos hasta el momento.

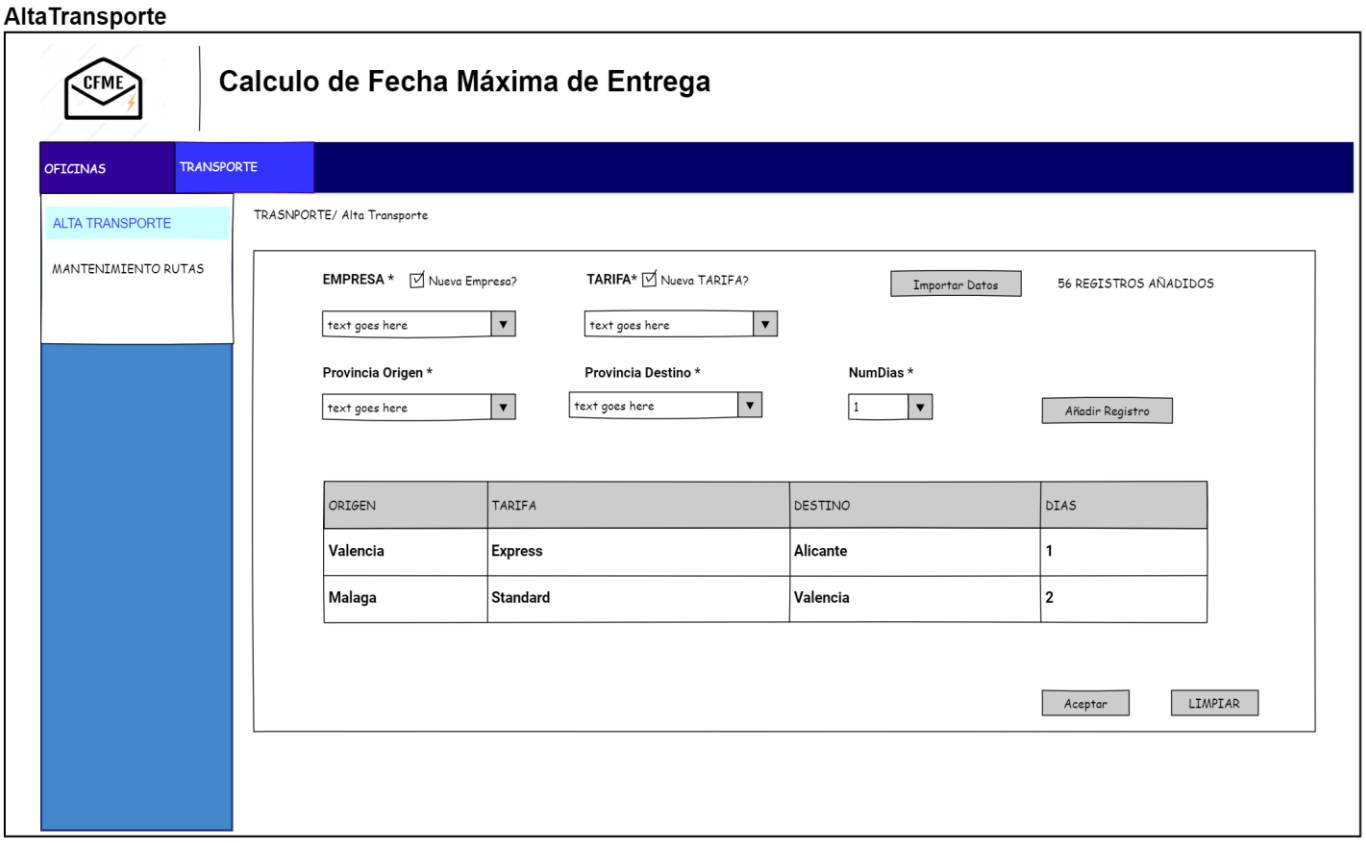



*Ilustración 5 Diseño Web: Alta Transporte*

En la Ilustración 7 se muestra el diseño de la ventana de mantenimiento de las tarifas. Dicha ventana la usarán las empresas que requieran de la generación de fechas de entrega para editar los datos de sus empresas, en concreto, la relación de sus oficinas con las tarifas de las empresas de transporte.

En este caso, la ventana contiene un filtro a través del cual el usuario puede buscar por su empresa (campo obligatorio), descripción de la oficina y tarifa a la que está suscrita. Una vez rellenados los campos obligatorios el usuario puede buscar o limpiar.

Si hace clic en buscar el sistema mostrará en la tabla de abajo los registros comunicados por el servidor que coinciden con los campos especificados. Si hace clic en limpiar el sistema eliminará la información introducida en todos los campos.

Los registros mostrados en la tabla podrán ser modificados, variando algún campo que lo forman o eliminados para que no vuelvan a aparecer en la base de datos. Cualquiera de estas dos acciones mostrará al usuario la advertencia de persistencia de datos y comunicará el resultado de la operación.

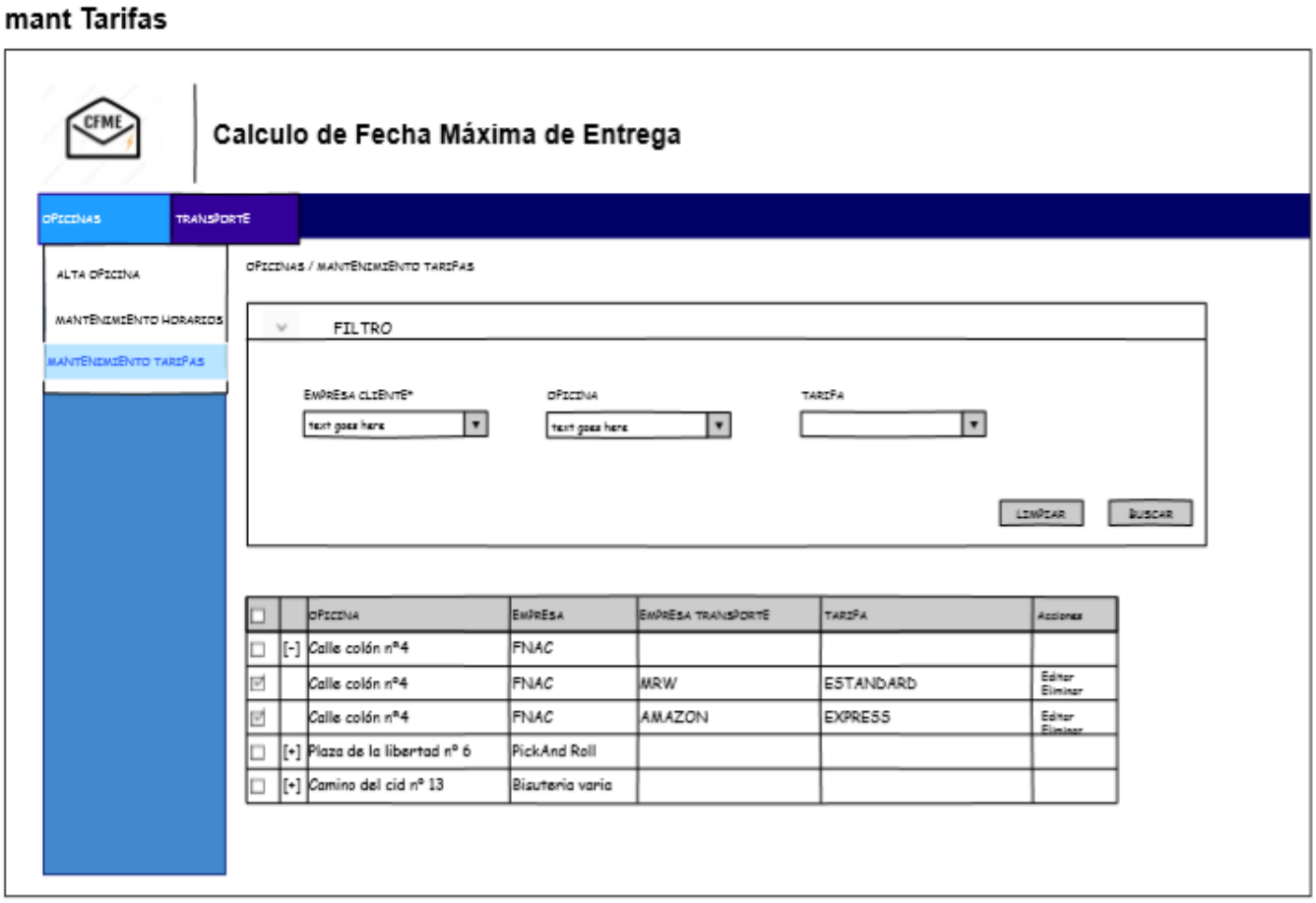



*Ilustración 6 Diseño Web: Mantenimiento Tarifas*

En la Ilustración 7 se muestra el diseño creado para dar de alta oficinas en el sistema. A través de esta interfaz el usuario puede crear una nueva empresa y añadir una oficina a ella marcando el *checkbox* o, en caso contrario, añadir una oficina a una empresa ya dada de alta con anterioridad.

Cuando el usuario crea o selecciona el nombre de su empresa debe seleccionar una o varias tarifas de una empresa de transporte. Para ello se desarrollará un *treeselect* que organizará las diferentes empresas de transporte con sus tarifas ofertadas. Además, se deberá indicar el código postal en el que se ubica la oficina y sus horarios. Para ello el sistema muestra una tabla donde el usuario puede seleccionar distintos horarios para su oficina y darlos de alta simultáneamente.

Una vez se han rellenado todos los campos de la interfaz si el usuario puede hacer clic en aceptar el sistema mostrará una ventana de advertencia de persistencia de datos y si el usuario acepta el riesgo se enviará una petición de alta de los datos recopilados, haciendo que el sistema muestre el resultado de la operación comunicada por el servidor. En cambio, si el usuario hace clic en limpiar el sistema borrará todos los datos introducidos en la pantalla.

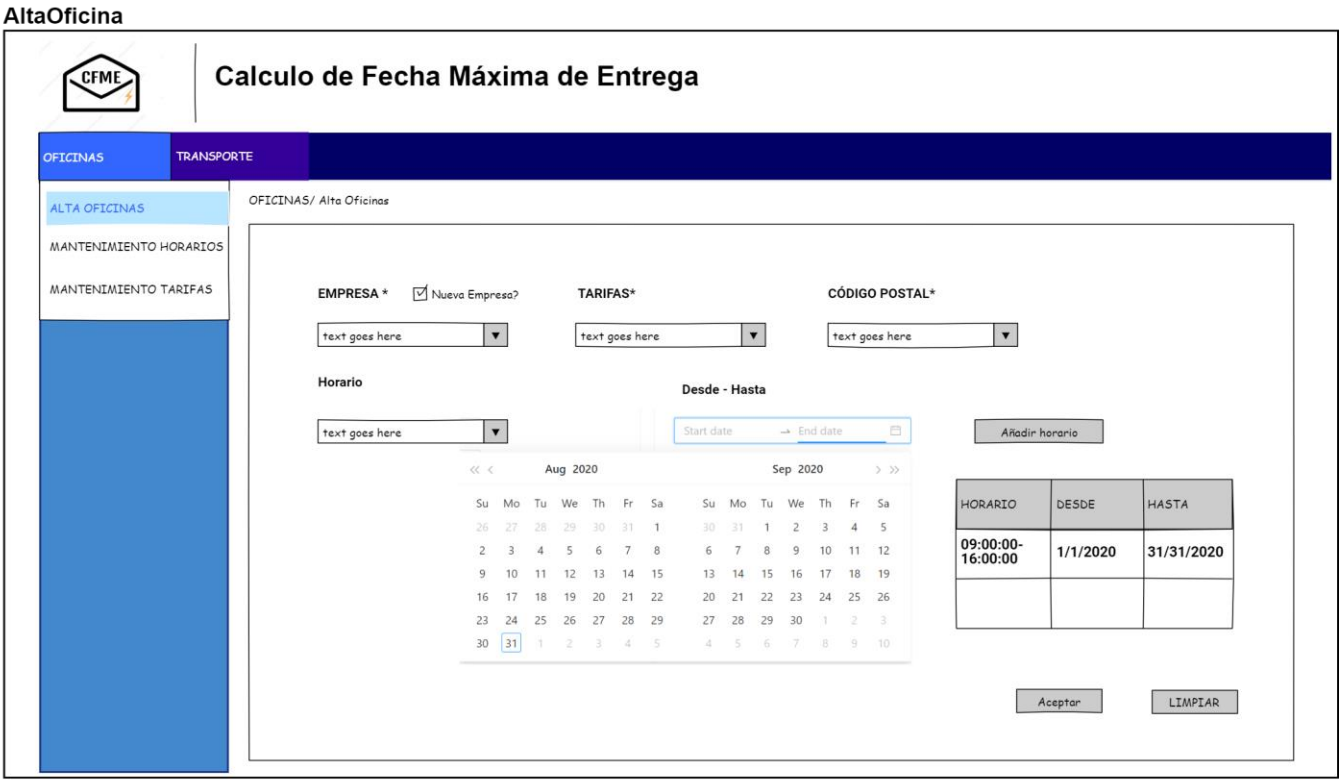



*Ilustración 7 Diseño Web: Alta Oficina*

Y, por último, la tercera pantalla de edición de datos es la ventana de mantenimiento de horarios mostrada en la Ilustración 8 . Esta pantalla será utilizada por usuarios de empresas que requieran del servicio web . A través de ella el usuario puede modificar los datos de las oficinas que ha dado de alta con anterioridad. Para ello debe rellenar los campos obligatorios del filtro y darle a buscar, seleccionar el registro que desea y modificar los datos como en las anteriores pantallas.

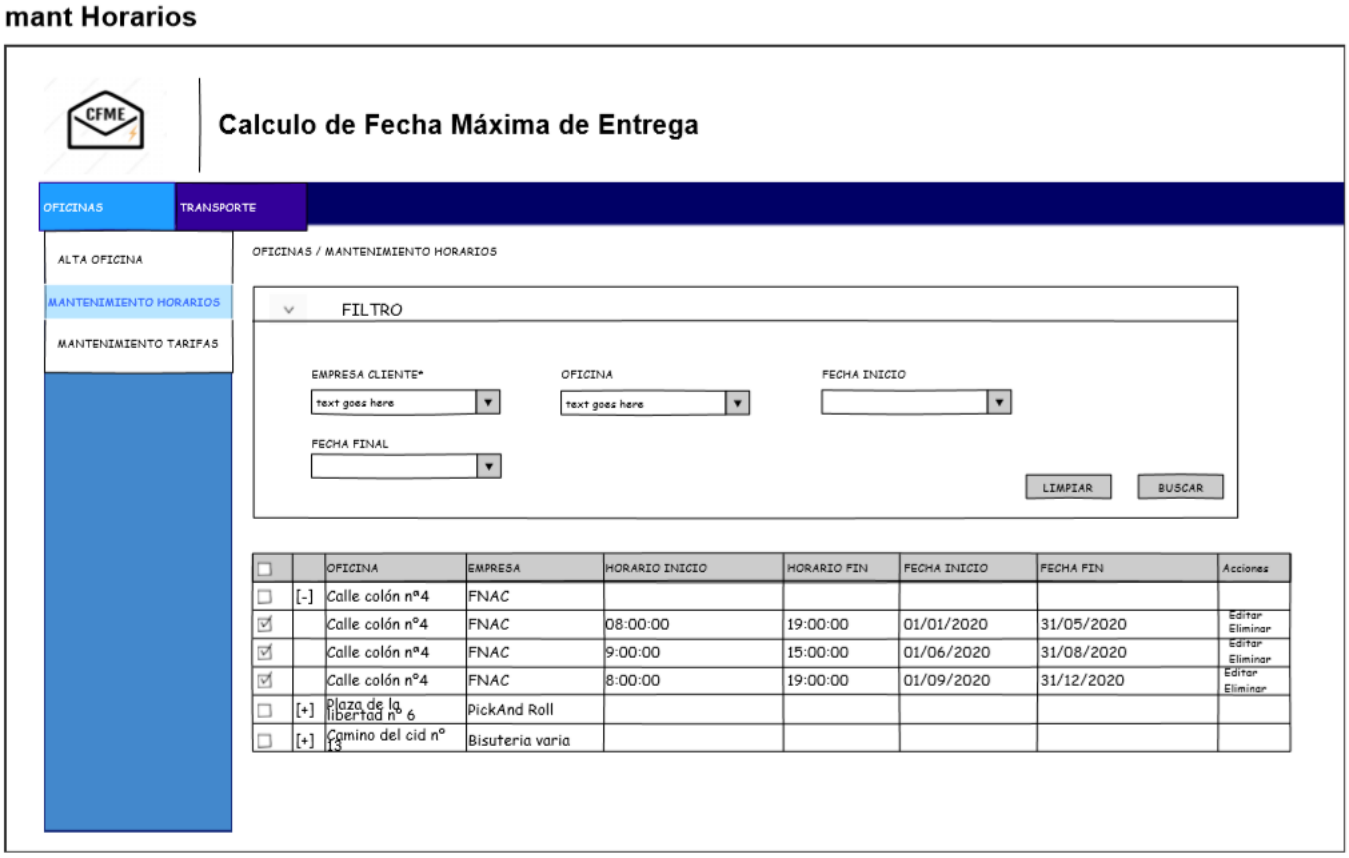



*Ilustración 8 Diseño Web: Mantenimiento Horarios*

### 4.2.3 Arquitectura de la base de datos

Una vez mostrado el diseño de la aplicación web y todo lo relacionado con el servidor CFME_FROTN podemos pasar al siguiente, el CFME_BACK. Este será un servidor JBoss 7.2 eap[8] ya que presenta una compatibilidad óptima con desarrollos implementados con las tecnologías empleadas en este proyecto. Además, su interfaz intuitiva y su alta capacidad de integración con el entorno de desarrollo IntelliJ, facilita el mantenimiento y la configuración del servicio web.

En el servidor CFME_BACK se encuentra desplegado el programa CFME.ear, una aplicación que implementa un servicio web, siguiendo la organización REST, utilizando la tecnología Spring Framework. Dicho servicio estará documentado y presentará una interfaz gráfica implementada a través de la librería Swagger2.

[8] https://access.redhat.com/articles/4901051

La base de datos es una relacional sobre la cual se ha implementado un modelo de datos para el almacenamiento y gestión de estos. Se ha elegido una base de datos de Oracle debido a la gran cantidad de documentación publicada a causa de su popularidad, hecho que facilita la integración de la base de datos con el servidor CFME_BACK. Para dicha integración se va a utilizar MyBatis, una herramienta de persistencia JAVA.

El modelo de datos diseñado para el servicio es el mostrado en el Apéndice 1. Dicho modelo permite el almacenamiento de distintas empresas de transporte con diferentes tarifas que pueden ser contratadas por las empresas interesadas. Cada empresa cliente puede registrar diferentes oficinas en distintas localidades, identificadas por el código postal, además, cada oficina puede tener distintos horarios. Las tarifas de transporte comprenden distintos trayectos entre provincias y asignan un número de días para cada uno de ellos. Por último, los días festivos están identificados por una fecha y un código de provincia, que mostrará la provincia afectada por el ámbito de dicho festivo. En caso de ser local, provincial o autonómico contendrán el número de provincia afectado, en caso de ser nacional no habrá provincias señaladas porque todas se verán afectadas.

### 4.2.4 Diseño del servicio

Una vez presentado el modelo de datos se mostrará el diseño de los mensajes para comunicarse con el servicio a implementar. Estos mensajes se enviarán a través del protocolo HTTP estandarizado y universal, utilizando el método POST. Para la descripción del mensaje, tanto de petición como de respuesta, vamos a utilizar el estándar de texto plano JSON. Y los diseños de petición y respuesta están definidos y documentados en la API empleada para la implementación del servicio.

La petición de entrada al servicio, mostrada en la Ilustración 9, contie cinco parámetros, de los cuales solo cuatro serán aceptados por el servicio. El parámetro codPostalDestino, codTarifaTransporte y fecAdmision son obligatorios, pero entre los parámetros codPostalOrigen y codOficina se debe de elegir uno.

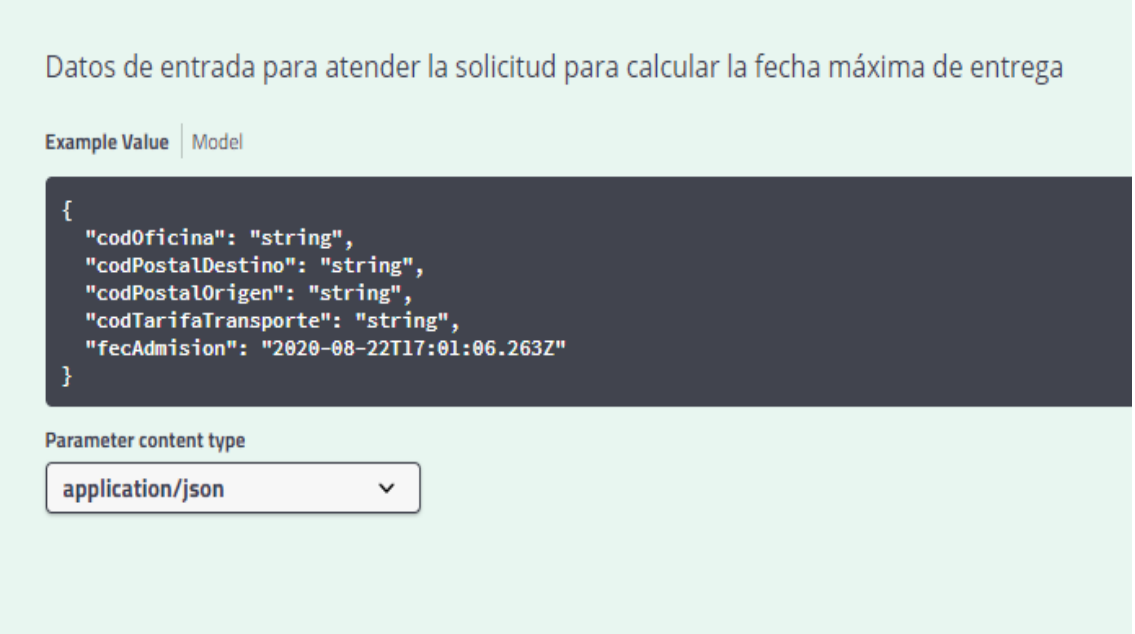



*Ilustración 9 Parámetros de la Petición*

La petición debe contener un origen, si se trata de una oficina debera proporcionar su código para poder calcular la fecha de entrega teniendo en cuenta su horario. En cambio, si se trata de un particular a través del codigo postal origen se podrá generar una fecha máxima de llegada del paquete, suponiendo que el pedido se genera en el momento y no depende de un horario de oficina.

Por ejemplo, una petición desde una oficina de una empresa de productos de peluqueria que oferta productos vía web y permite entre envio estandar y premium podría ser: el codigo de oficina referente a la susodicha, la tarifa premium si el cliente ha pagado la suscripción o estandar en caso contrario, el codigo postal destino es facilitado por el cliente a la hora de hacer el pedido y la fecha de admisión es el dia de hoy que se puede obtener a partir de varias funciones lógicas.

Una vez presentado y explicado el diseño de la petición pasamos a ver el diseño de los diferentes tipos de respuesta implementados para el servicio. Se ha tomado una captura de la interfaz implemntada para una mejor descripción de la respuesta y se muestra en la Ilustración 10.

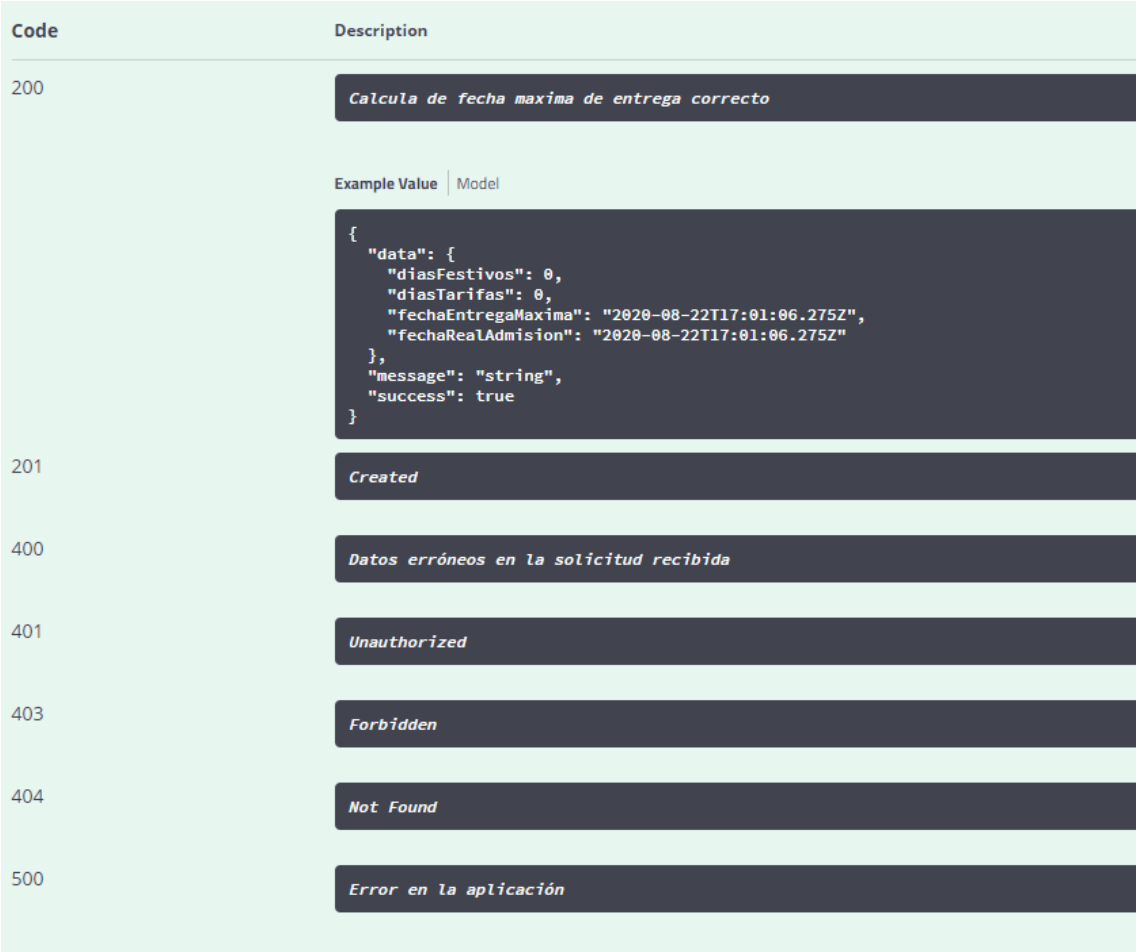



*Ilustración 10 Parámetros de la Respuesta*

Se puede observar que hay distintos tipos de respuesta encabezados por un código que sigue el estándar del protocolo HTTP. Entre ellos cabe destacar el código 200 que indica que la petición es correcta y la fecha de entrega máxima se ha calculado correctamente. Bajo de dicho código, se encuentra un modelo de la respuesta que permite saber los campos que la forman para poder trabajar con ella de antemano. El resto de los códigos indican que la petición no se ha generado y que ha habido un problema en la realización de la petición, o en caso del 500 que ha habido un problema en la aplicación durante la generación de la fecha.

El modelo de la respuesta correcta está formado por los campos: diasTarifa que informa sobre el número de días que tarda el trayecto desde la provincia origen hasta el destino; diasFestivos que muestra el número de días que se demora la entrega debido al carácter no laboral de estos; fechaRealAdmision, que indica la fecha en la cual la oficina tramita el pedido y no cuando se realiza el pedido; fechaEntregaMaxima es la fecha límite a la que se compromete a entregar su pedido el ofertante.

### 4.2.5 Diseño del algoritmo

Para terminar, vamos a mostrar el algoritmo implementado para la generación de fechas de entrega máxima de paquetes. Para ello en la Ilustración 11 se muestra el diagrama de flujo que sigue el servicio web.

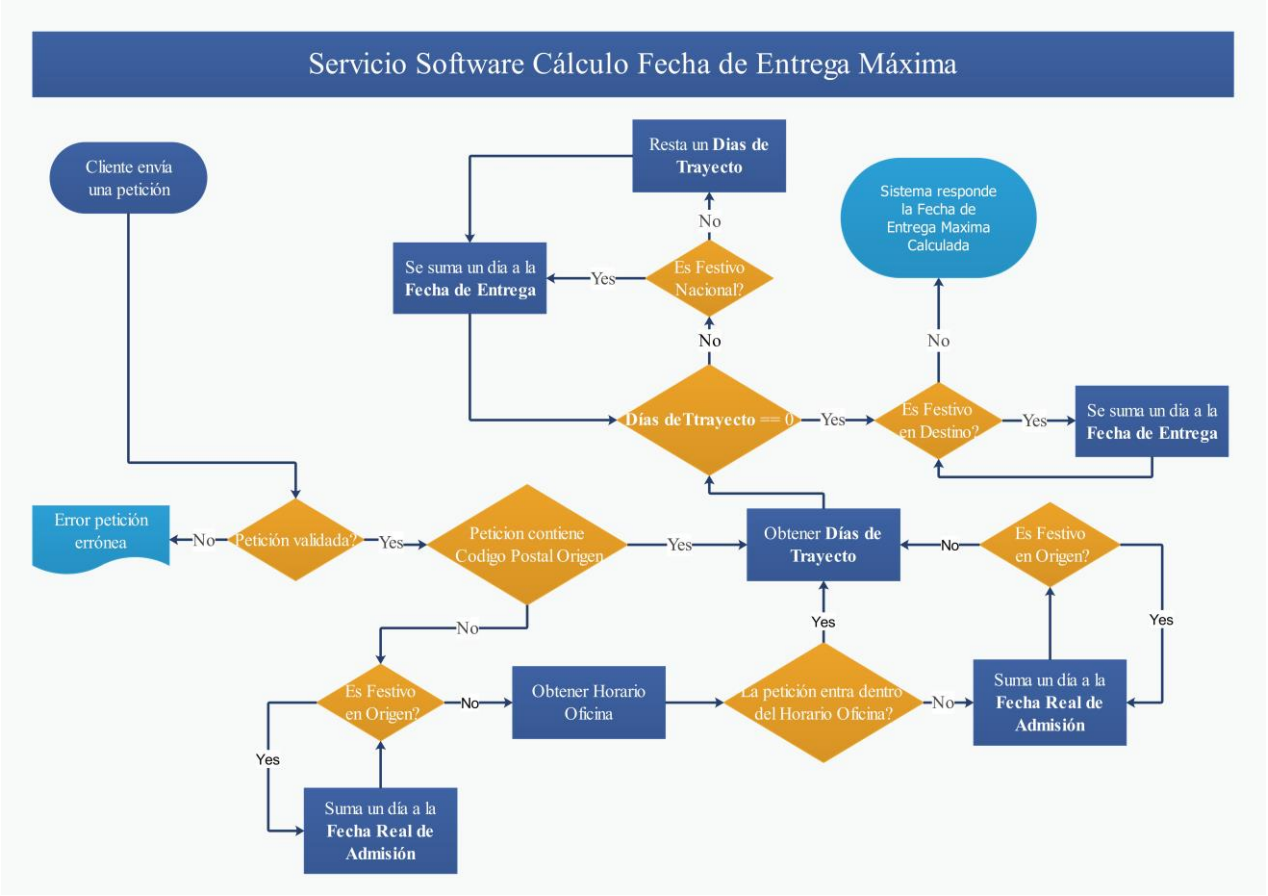



*Ilustración 11 Flujo del Servicio Software Cálculo Fecha de Entrega Máxima*

El sistema calcula primero *la fecha real de admisión*, es decir, la fecha en la que la oficina gestiona el pedido y no la fecha en la que se envía la petición. Esto permite al sistema generar una fecha de entrega acorde con los horarios activos de las oficinas y los días festivos en origen. Por ejemplo, si la petición se tramita un sábado a media noche el sistema detectará que la oficina está cerrada y que al siguiente es domingo por lo que la fecha real de admisión será el próximo lunes.

Una vez generada la fecha real de admisión, el sistema pasa a recuperar desde la base de datos los días que tarda el transporte del paquete, teniendo en cuenta la tarifa seleccionada por el cliente y el código postal destino. Durante estos días cabe tener en cuenta que pueden ser festivos durante el transporte, por lo que el sistema debe ir contando a partir de la fecha real de admisión, día por día si es festivo o no. En caso de ser festivo no debe consumir días de transporte, pero debe sumar un día más por festivos durante el transporte.

Por último, una vez conocida la fecha en la que llegará el paquete a su lugar de entrega el sistema comprobará si la fecha es festiva en el destino, de ser así el sistema sumará dichos días.

Por tanto, la fórmula que se seguirá para la generación de fechas de entrega máxima de paquetes es la siguiente.

**Fecha de entrega máxima de paquetes = Fecha real de admisión + Días de transporte + Días festivos utilizados**

**Días festivos utilizados = Festivo en destino + Festivos durante el transporte**

**Fecha real de admisión = Días por horario de oficina + Festivos en origen**

## 4.3 Tecnología utilizada

En el siguiente apartado se mostrará detalladamente todas las tecnologías que se han empleado para la realización del proyecto y se indicará su uso en él. Primero se presentarán las aplicaciones de escritorio que conforman el entorno de trabajo y se continuará con las tecnologías para las funcionalidades específicas del proyecto.

### 4.3.1 Oracle SQL Developer Data Modeler

Esta aplicación es una herramienta de diseño de modelos de datos que permite al usuario implementar un modelo de manera gráfica e intuitiva. Además, una vez diseñado el modelo, el usuario puede exportar

su diseño y la aplicación genera un script en el cual genera el modelo en la base de datos.

Esta aplicación es una herramienta de diseño de modelos de datos que permite al usuario implementar un modelo de manera gráfica e intuitiva. Además, una vez diseñado el modelo, el usuario puede exportar su diseño y la aplicación genera un script en el cual queda definida dicho diseño. Esto permite al usuario ejecutar el script directamente sobre su gestor de bases de datos obteniendo así una arquitectura de base de datos idéntica a la diseñada.

### 4.3.2 Oracle SQL Developer

Oracle SQL Developer es un entorno de desarrollo integrado para trabajar con SQL en bases de datos Oracle. Dicho entorno permite crear, modificar y consultar la base de datos. Esta aplicación se puede obtener de forma gratuita desde la web oficial de Oracle.

Esta tecnología ha sido elegida por su popularidad y fácil acceso que hacen de ella una aplicación muy documentada en la web. Además, es una de las utilizadas durante el grado cursado y es muy empleada en las empresas para la gestión de sus datos.

### 4.3.3 IntelliJ IDEA

IntelliJ IDEA es un entorno de desarrollo integrado para el desarrollo de programas informáticos. Esta desarrollado por JetBrains y se puede obtener mediante su compra en la página oficial. Es uno de los entornos de desarrollo más completos.

IntelliJ IDEA ha sido elegido para el desarrollo de este proyecto gracias a la universidad que oferta de manera gratuita una licencia de estudio. Además, es muy intuitivo y dispone de una amplia variedad de funciones que otros entornos carecen como la integración de la base de datos en el mismo.

### 4.3.4 Postman

Es una aplicación que permite generar y lanzar peticiones HTTP contra servidores. Ofrece una interfaz intuitiva a la hora de añadirle campos de cabecera y parámetros en la petición, así como para un cuerpo del mensaje con formato determinado. Además, permite guardar las consultas realizadas.

Es muy utilizada a la hora de hacer pruebas en el momento de desarrollo para comprobar el correcto funcionamiento los servicios levantados.

### 4.3.5 GitHub

Esta herramienta ofrece un espacio en la nube, donde subir los ficheros que componen un proyecto. A este espacio se le llama repositorio, y permite tanto acceder a él desde diferentes dispositivos, como que otras personas puedan acceder a él.

Las acciones que se pueden realizar sobre un repositorio son las que define Git, proyecto al que GitHub amplia ofreciéndole una interfaz.

Estas acciones están diseñadas para facilitar el control de versiones. Uno de los casos de uso más habituales de esta gestión de versiones es el de mejorar o refinar la funcionalidad de un proyecto que se encuentra desplegado en producción. Para evitar que un fallo no previsto desencadene una caída del servicio, lo frecuente es copiar el repositorio en un ordenador ajeno al entorno de producción y hacer las modificaciones y prueba necesarias, de modo que cuando se esté seguro del funcionamiento, fusionarlo con lo que se encuentra en el repositorio, generando una nueva versión que será desplegada en producción.

### 4.3.6 JBoss EAP 7.2

Esta herramienta es una plataforma de ejecución de aplicaciones web basada en un servidor de aplicaciones usada para la creación, despliegue y mantenimiento de aplicaciones JAVA. Esta plataforma se puede adquirir de forma gratuita en la web oficial de Red Hat.

Esta plataforma ha sido seleccionada por su fácil configuración, su alta compatibilidad con aplicaciones java y su arquitectura orientada a servicios web que hacen de este servidor perfecto para albergar nuestro proyecto.

### 4.3.7 Docker

Docker facilita el despliegue de una aplicación dejándola aparte del sistema operativo de la máquina en la que se encuentra. De esta tecnología aparece el concepto de contenedor, que es una imagen configurada con anterioridad con lo necesario para que la aplicación funcione.



Docker implementa esta funcionalidad y permite su fácil configuración a través de Docker ToolBox, un paquete de aplicaciones que permite obtener y acceder fácilmente a las imágenes preconfiguradas que necesites. Esta tecnología aporta la herramienta Oracle VM, Docker QuickStrat terminal y el Kitematic en fase Alpha, que simplifica el uso de los contenedores.

### 4.3.8 MyBatis

MyBatis es una herramienta de persistencia Java que se encarga de mapear sentencias SQL y procedimientos almacenados con objetos a partir de ficheros XML o anotaciones.

Esta herramienta ha sido seleccionada para el proyecto porque tiene una fácil integración con Spring y ha sido utilizada durante muchos años, por lo que hay mucha información en la nube.

Además, ha sido el *framework* con el que el autor del documento ha estado trabajando todo el periodo de prácticas y como las sentencias se escriben en XML el proyecto abarca varios tipos de ficheros que se utilizan en la integración de aplicaciones (JSON y XML).

### 4.3.9 Swagger-ui

Es una API diseñada para simplificar el desarrollo a equipos, usuarios y empresas. Es de uso libre y sirve de unión entre la parte de *back-end* de los servicios y *front-end*.

El empleo de esta API ayuda a que, en tiempo de desarrollo, el servicio quede documentado y sea fácil de comprender. Además, es inevitable que el servicio sea modificado o mejorado, por ello esta API ayuda a que no sea tan tedioso modificar la documentación del servicio ya que se genera simultáneamente que en el desarrollo.

### 4.3.10 Spring

Es un gran *framework* que recoge todas las librerías necesarias para desarrollar una aplicación web con JAVA. Requiere una configuración sencilla para el empleo en desarrollos, permite al programador indicar mediante anotaciones los objetos, también conocidos como *Beans*, que van a ser usados por otros componentes de la aplicación, de manera que Spring automatiza la gestión de la instanciación y la paralelización.

La librería más importante empleadas para este proyecto han sido RestController que permite exponer URIs REST, que son necesarios para

la ejecución de los procesos de la aplicación. Permite además que operaciones va a admitir nuestro servicio, que parámetros van a ser necesarios para su uso y que objetos va a devolver en formato JSON, siguiendo el estándar HTTP.

### 4.3.11 Maven

Es una herramienta de código libre, con licencia de Apache Software que se desarrolló en un principio para automatizar el procesa de compilación de proyectos basándose en ficheros con extensión XML.

Su funcionamiento se basa en ficheros `pom.xml` que recogen los diferentes módulos que componen el proyecto, así como versiones y repositorios donde descargarlos.

Otra funcionalidad importante que suministra esta herramienta es el ciclo de vida que son las fases o etapas a las que Maven somete el proyecto. Las usadas en este proyecto son la compilación, la prueba o test, y el empaquetado.

### 4.3.12 REACT

Es una librería que permite crear aplicaciones hecha para trabajar con aplicaciones tanto robustas como sencillas y al ser declarativa hace fácil seguir patrones de diseño y crear interfaces de usuario interactivas de manera eficiente.

REACT es muy eficiente ya que los cambios que impactan sobre el DOM o el HTML hay que volver a renderizarlo, pero REACT permite realizar el cambio en ese único elemento. Además, esta librería funciona utilizando Babel que permite utilizar las últimas características de JavaScript en cualquier navegador, aunque no las soporte.

Ha sido seleccionado por su estructura en componentes, que permiten dividir aplicaciones complejas en pequeñas pizas de código encapsuladas sencillas que permiten mantener el programa limpio y sencillo.

# 5. Desarrollo de la solución

Habiendo presentado los diseños y las tecnologías utilizadas, se mostrará, a continuación, como se ha ido desarrollando el proyecto paso a paso.

## 5.1 Configuración del entorno

El desarrollo comienza con la configuración completa del entorno de trabajo, esto es, la instalación de todas las herramientas de escritorio que se utilizarán durante la implementación del proyecto. La instalación de todo el software necesario no es compleja ya que las tecnologías a utilizar están bien documentadas en sus webs y se encuentran definidas en el apartado 4.3.

Una vez todo está instalado en nuestro entorno pasamos a configurar nuestra base de datos, mediante la tecnología Docker se descarga una imagen Docker que lleva instalado Oracle-xe-11g. Esto nos permite almacenar los datos que sean necesarios en un espacio virtual ya configurado para su uso. Para ello se debe iniciar la aplicación Docker Quickstart Terminal y copiar el siguiente comando, como se muestra en la Ilustración 12 : 'docker pull wnameless/oracle-xe-11g-r2'.

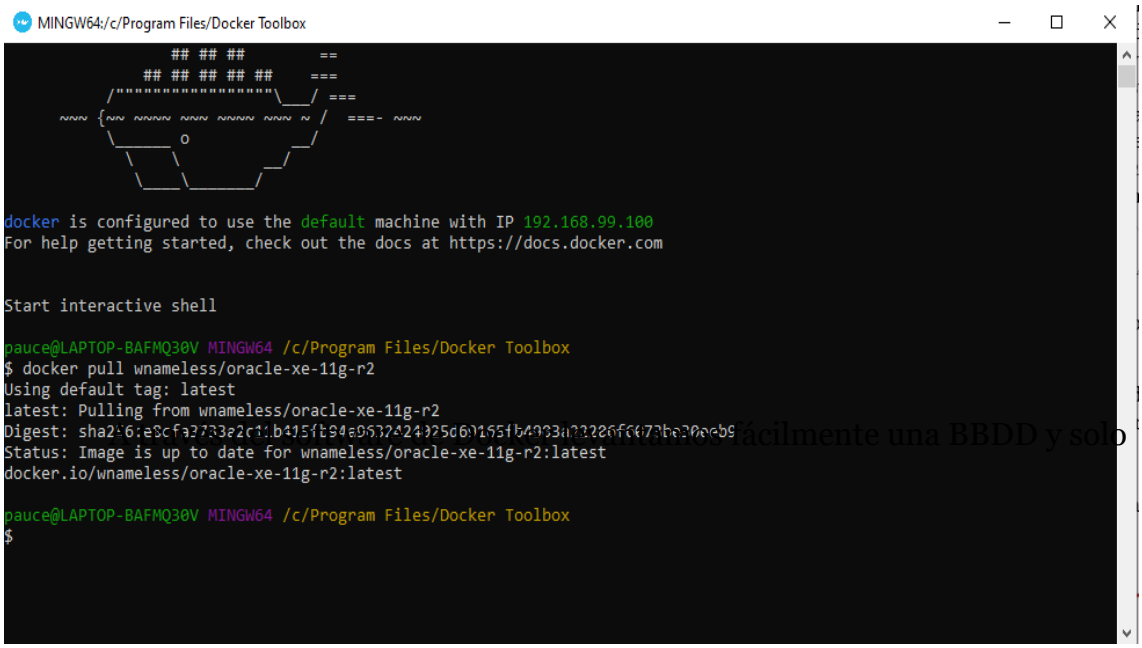



*Ilustración 12 Consola de Docker*

Cuando la imagen ya ha sido descargada se debe crear un contenedor mediante el comando 'docker run -d -p 1521:1521 -e ORACLE_ALLOW_REMOTE = true wnameless/oracle-xe-11g-r2'.

El siguiente paso será el mostrado en la Ilustración 13 que consiste en abrir los puertos de la máquina virtual para poder acceder a ella desde el ordenador. Para ello inicia OracleVirtualbox y sobre la máquina *default* pulsa el botón derecho y selecciona la opción de 'Configuración'.

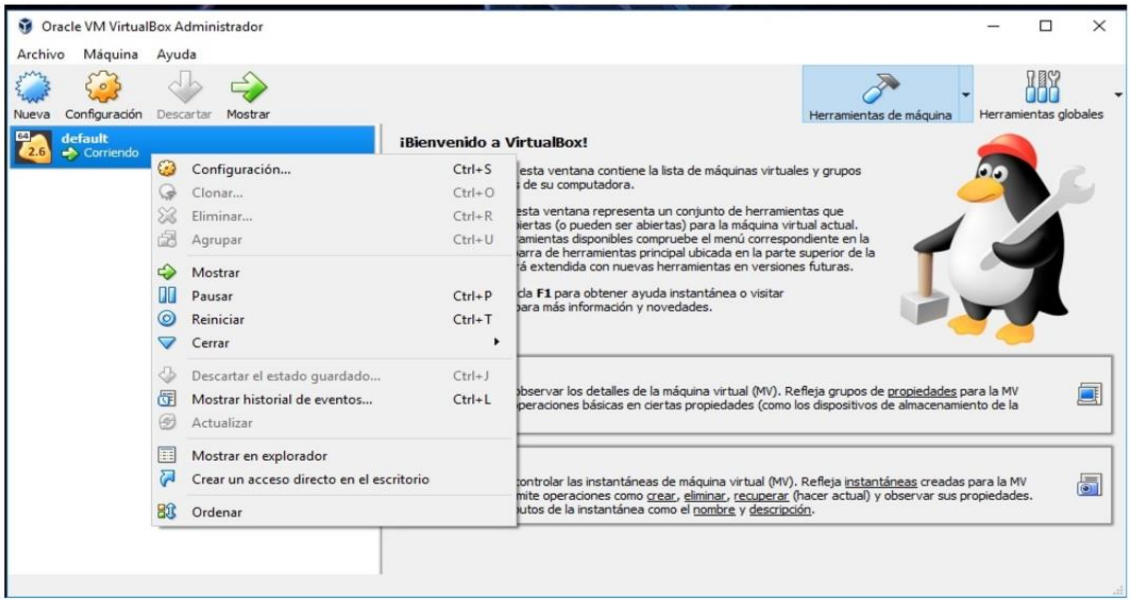



*Ilustración 13 Oracle VirtualBox Configurar Puertos 1*

Al desplegar las opciones avanzadas y pulsar sobre el botón de reenvío de puertos se mostrará una pantalla como la mostrada en la Ilustración 14. En dicha pantalla se debe crear una nueva regla de reenvío de puertos estableciendo el valor de ambos a '1521'.

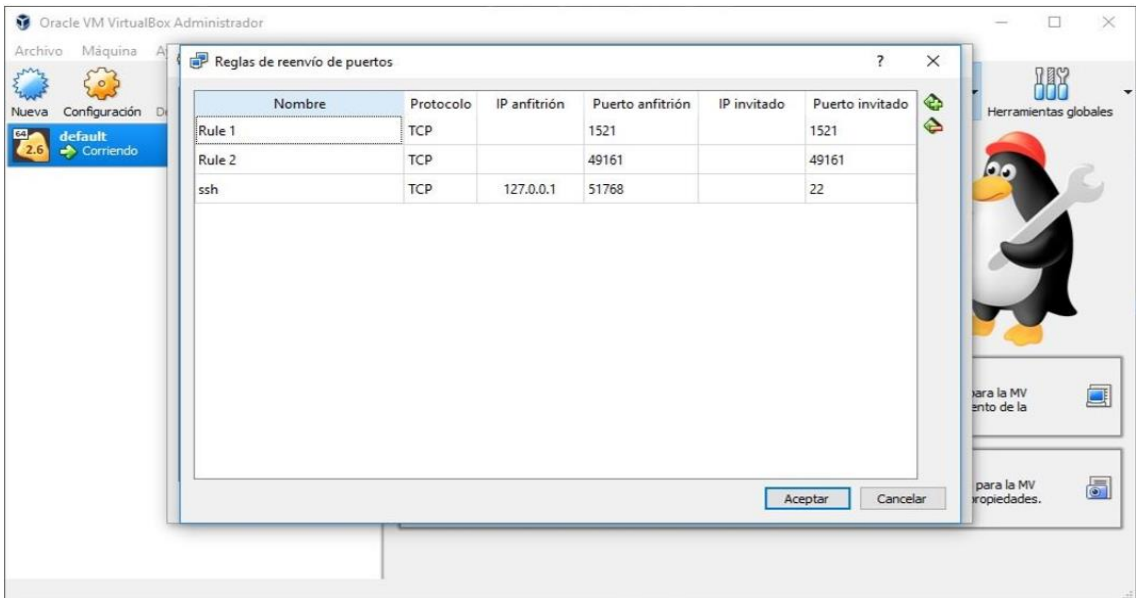



*Ilustración 14 Oracle Virtual Box Configurar Puertos 2*

Tras realizar el cambio en nuestra máquina virtual, se debe probar la conexión entre ella y el software de gestión de bases de datos previamente instalado, en este caso sqldeveloper mostrado en la Ilustración 15. Para ello se debe añadir una nueva conexión introduciendo los datos que aparecen en la ilustración 16 con usuario **system** y contraseña **Oracle**. Para comprobar si tenemos acceso desde nuestro ordenador al contenedor creado se debe hacer al botón probar y el sistema informará si la conexión ha sido exitosa, en caso contrario revisar los dos pasos anteriores.

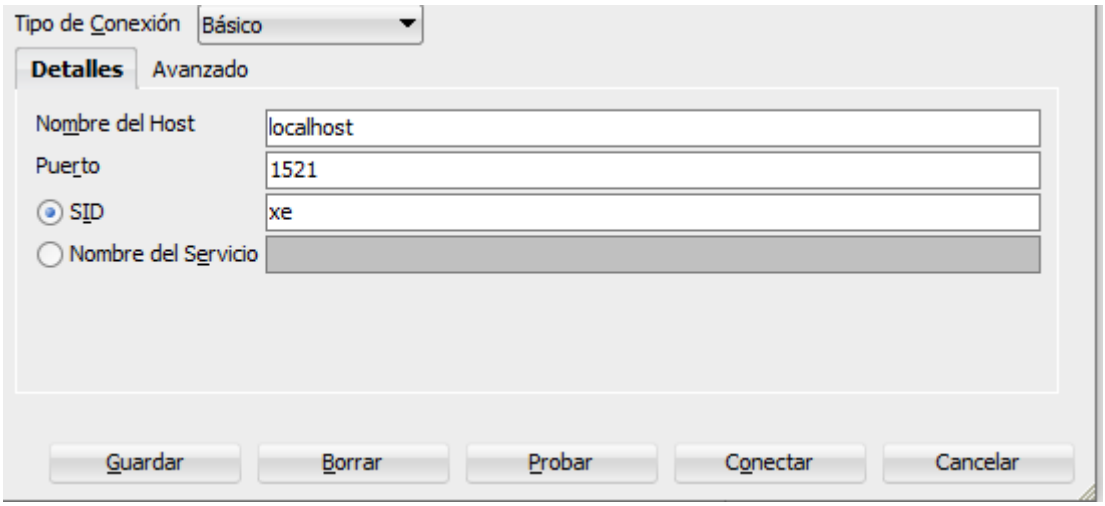



*Ilustración 15 SqlDeveloper Configurar Conexión a Base de Datos*

El siguiente paso va ligado con el diseño, ya que el software utilizado para implementar el modelo de datos, Datamodeler, puede generar un script a partir del modelo diseñado exportándolo con extensión `.ddl`, simplificando de esta manera la creación del esquema en la base de datos. Así, solo hace falta crear un usuario, un *tablespace* y garantizar los permisos al nuevo usuario para ejecutar cambios en la base datos. Para ello simplemente se debe abrir el fichero con extensión `.ddl` en la aplicación sqldeveloper y hacer clic sobre ejecutar script.

En este punto, con el esquema relacional implementado, se puede comenzar a insertar los datos necesarios para el correcto funcionamiento de nuestra aplicación. Para ello utilizamos fuentes de datos abiertas que suministran todos los códigos postales de España y sus ubicaciones en las distintas provincias. Por otro lado, los festivos relacionados con sus ubicaciones han sido obtenidos por la administración del gobierno

(https://administracion.gob.es/) y dicha tabla tendrá que ser actualizada año a año ya que solo contiene los festivos del año 2020. Además, para las claves primarias de algunas tablas se han implementado secuencias, simplificando así la inserción de grandes volúmenes de datos.

A partir de aquí la base de datos ya está implementada y podemos pasar a configurar lo necesario para la implementación de la lógica. Para ello deberemos configurar en primera instancia nuestro servidor, enlazarlo con la base de datos e integrarlo con nuestro entorno de desarrollo.

Primero descargamos el servidor de aplicaciones JBoss EAP 7.2 de su web oficial y modificamos su archivo "standalone.xml" que se encuentra en la ruta "C:\Program Files\jboss-eap-7.2\standalone\configuration" para permitir al servidor comunicarse con nuestra base de datos. Para ello debemos definir en la etiqueta "<datasources>" la cadena de conexión a la base de datos con el usuario de acceso como se muestra en la Ilustración 16.

```
<xa-datasource jndi-name="java:/jdbc/calculadorXADS" pool-name="calculadorXADS">
    <xa-datasource-property name="URL">
        jdbc:oracle:thin:@127.0.0.1:1521/xe
    </xa-datasource-property>
    <xa-datasource-class>oracle.jdbc.xa.client.OracleXADataSource</xa-datasource-class>
    <driver>oracleXA</driver>
    <xa-pool>
        <is-same-rm-override>false</is-same-rm-override>
        <no-tx-separate-pools>true</no-tx-separate-pools>
    </xa-pool>
    <security>
        <user-name>CALCULADOR_OWN</user-name>
        <password>CALCULADOR_OWN</password>
    </security>
    <validation>
        <valid-connection-checker class-name="org.jboss.jca.adapters.jdbc.extensions.oracle.OracleValidConnectionChecker"/>
        <background-validation>true</background-validation>
        <stale-connection-checker class-name="org.jboss.jca.adapters.jdbc.extensions.oracle.OracleStaleConnectionChecker"/>
        <exception-sorter class-name="org.jboss.jca.adapters.jdbc.extensions.oracle.OracleExceptionSorter"/>
    </validation>
</xa-datasource>
```

*Ilustración 16 Configuración de Drivers JBoss 1*

Una vez declarada la cadena de conexión (IP/Puerto/Nombre del Servicio) para nuestra base de datos, solo falta instalar el controlador al servidor. En nuestro caso vamos a descargar el "ojdbc6" que se puede descargar desde la web oficial de Oracle. Lo siguiente es copiar el archivo descargado en la ruta 'C:\ ProgramFiles \ jboss-eap-7.2 \modules \system \layers \base \com \oracle\main' y declararlo en la etiqueta "<drivers>" del archivo 'standalone.xml' mostrada en la Ilustración 17. En caso de que dicha ruta no exista es conveniente crearla para almacenar en ella los distintos drivers a utilizar.

```
<drivers>
    <driver name="h2" module="com.h2database.h2">
        <xa-datasource-class>org.h2.jdbcx.JdbcDataSource</xa-datasource-class>
    </driver>
    <driver name="oracleXA" module="com.oracle">
        <driver-class>oracle.jdbc.driver.OracleDriver</driver-class>
        <xa-datasource-class>oracle.jdbc.xa.client.OracleXADataSource</xa-datasource-class>
    </driver>
</drivers>
```

*Ilustración 17 Configuración de Drivers JBoss 2*

En este punto el servidor ya ha sido completamente configurado pasamos a integrarlo con nuestro en entorno de desarrollo, IntelliJ. Debemos seleccionar la pestaña de "Run" y hacer clic sobre la opción "Edit Configurations". Se nos abrirá una pestaña como la mostrada en la Ilustración 18 con un panel a la izquierda que arriba del todo tiene un símbolo de suma, hacemos clic sobre él. Se nos mostrará varias configuraciones de las cuales debemos seleccionar la de JBoss, en concreto Local. Una vez seleccionada, se abrirá una pantalla con varios campos los cuales deberemos rellenar con los datos referentes a nuestro entorno.

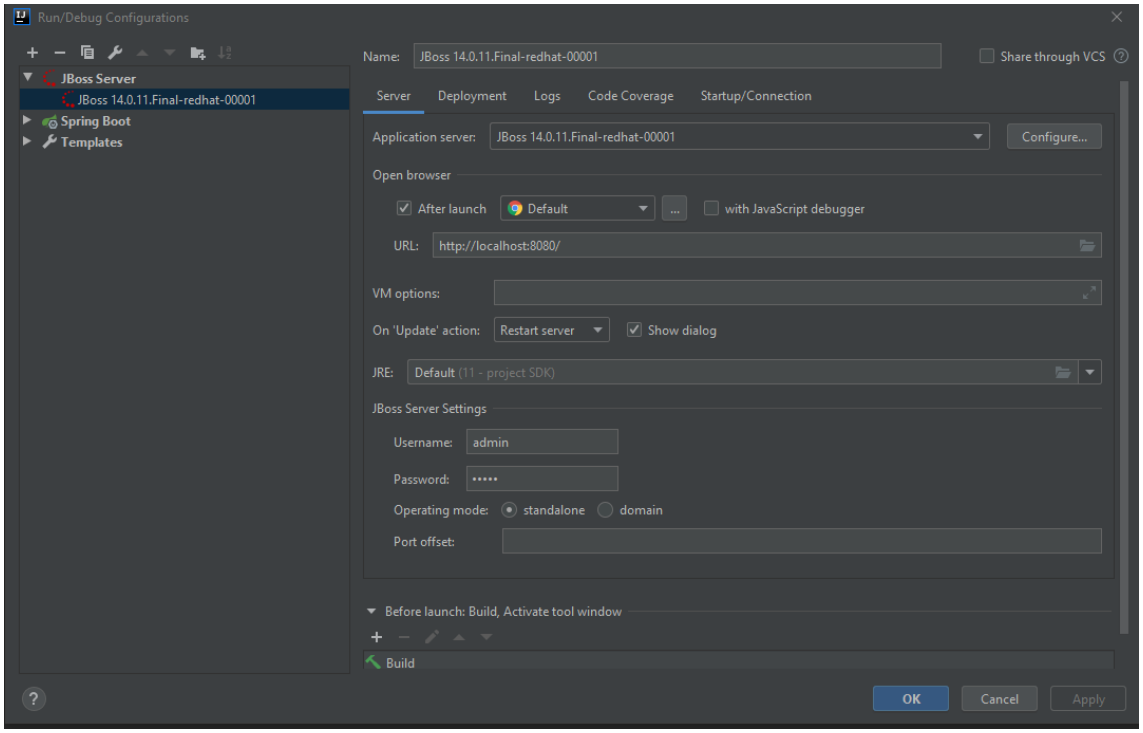



*Ilustración 18 Integración de servidor JBoss con IntelliJ*

## 5.2 Creación de la estructura Spring

Terminada la integración del servidor y habiendo configurado correctamente el acceso a la base de datos, nuestro entorno de desarrollo ha sido correctamente configurado, ahora se pasará a la creación de un nuevo proyecto desde cero.

Para crear el proyecto se va a utilizar la herramienta web Spring Initializr[9] que permite crear proyectos ya configurados para su uso, es decir, con Maven instalado y configurado para ser ejecutado. Así, generamos distintos proyectos que después se organizarán para conseguir una arquitectura estable, clara y fácil de escalar.

Mediante dicha web se montará un proyecto principal compuesto por distintos proyectos secundarios, es decir, se empleará la organización "Parent Pom" para facilitar la configuración del proyecto y hacerlo más intuitivo. Dicha organización parte de un proyecto padre que contiene fichero "pom.xml" que recopila toda la información común de los cinco proyectos que se encuentran dentro de él. Esto es útil a la hora de declarar las versiones de las librerías variables globales y otras constantes utilizadas en todos los procesos de integración como se muestra en la Ilustración 19. Además, se encuentran declarados los proyectos secundarios o hijos, que conforman el programa, y los *plug-in* que se van a emplear, como son el de apache Maven o MyBatis.

[9] https://start.spring.io/

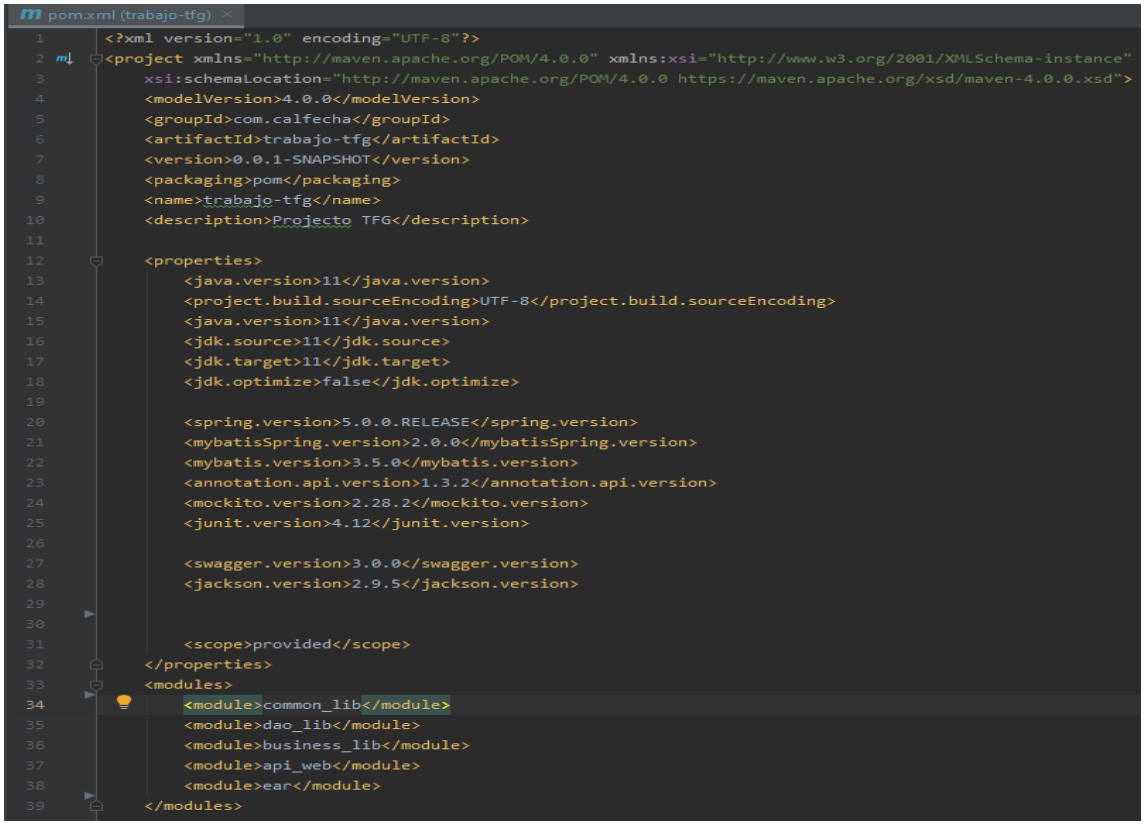



*Ilustración 19 Configuración Parent Pom*

En este punto, la estructura del código ya está construida, así que se puede empezar a declarar las dependencias o librerías que se van a utilizar. Esto se hace a través de Maven indicando en los ficheros 'pom.xml' de cada proyecto las librerías que se necesitarán. No hace falta ponerlas todas desde el principio ya que el *plug-in* de Maven detecta las dependencias en el código, de manera dinámica, y las añade al fichero automáticamente.

Después de haber importado las librerías principales y creado los módulos del proyecto se podría afirmar que la estructura software es estable y se puede comenzar a escribir código.

Después de haber importado las librerías principales y creado los módulos del proyecto se pasará a explicar cómo configurar una arquitectura sobre Spring para generar un servicio web.

Lo primero es necesario configurar el *AplicationContext interfaz*. Esto es, una subinterfaz de *BeanFactory* que será la encargada de manejar todos los objetos, también conocidos como los Spring *Beans* de nuestra aplicación. Se utilizará la organización definida por Kamlesh Kumar como "orientada a *Annotation-based* que mezcla la configuración vía XML con un conjunto de

etiquetas JAVA que se emplean en clases, métodos, constructores o campos para la configuración de los be*ans*. Algunas de estas anotaciones son @Component, @Service, @Controller…*"* [4].

En primera instancia hay que crear los archivos 'AplicationContext.xml' en cada uno de los proyectos hijo que componen el programa. En cada uno de estos archivos hay que indicar mediante la etiqueta "<contex:component-scan/>" los directorios donde se encuentran los archivos con las anotaciones JAVA y a través de la etiqueta <import/> importar los proyectos con los que están relacionados. Es decir, se debe indicar que el proyecto donde se alberga la capa de la interfaz se relaciona mediante llamadas con la capa de la lógica de negocio y que esta a su vez, se enlaza con el proyecto que contiene las llamadas a la base de datos. Esta estructura se puede observar en la Ilustración 20.

Por ejemplo el proyecto "api_web" contiene los servicios rest declarados con sus respectivas etiquetas(@RestController). Con la etiqueta anteriormente nombrada Spring escanea el directorio en busca de las etiquetas y crea los *beans* pertinientes. Además dicho archivo importa el fichero "aplicationContext-business.xml", montando así la estructura del programa.

Del mismo modo, el fichero "aplicationContext-business.xml" escanea el paquete donde se encuentran los servicios implementados en busca de la estiqueta "@Service" y por cada etiqueta econtrada crea un bean para las transacciones entre módulos. Además, este fichero importa el referente a la capa de acceso a la base de datos.

Este último fichero llamado "aplicationContex-Dao.xml" es diferente al resto ya que se trata del relacionado con la capa de acceso a base de datos. En él encontramos definido un *bean* de tipo "sqlSessionFactory", la etiqueta "<mybatis:scan>" que apunta al directorio donde se encuentran todos los ficheros de mapeo y otra etiqueta para indicar a través de que driver del servidor se va a acceder a la base de datos.

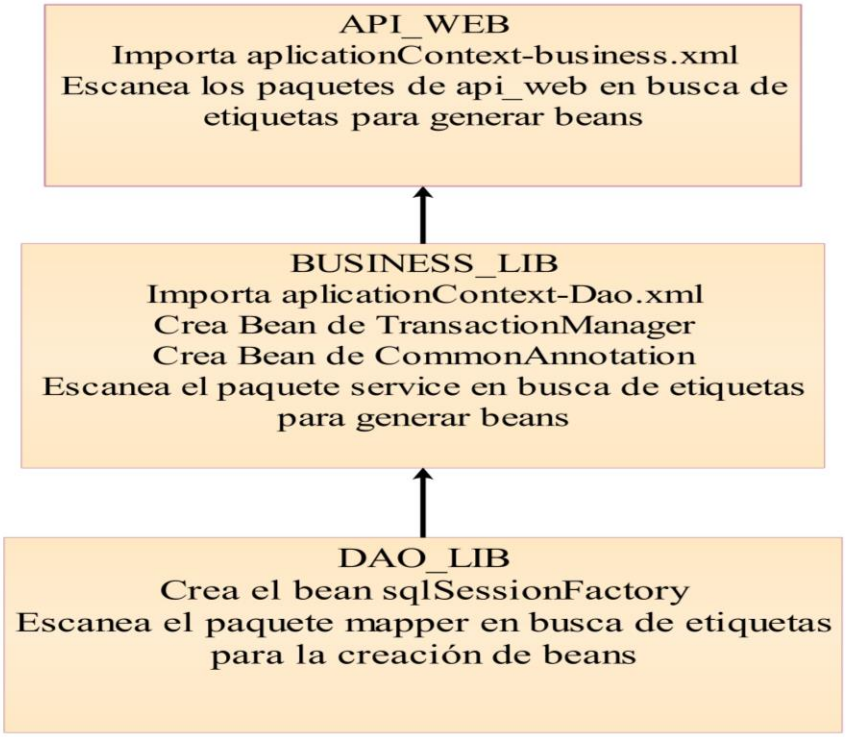



*Ilustración 20 Arquitectura SpringContext*

Una vez el *AplicationContext interfece* esté creado, solo falta configurar lo relativo al proyecto web. A partir de Spring Web MVC se generará un Servlet encargado de enviar peticiones a los diferentes manejadores de la aplicación. Configurando el fichero "web.xml" de la aplicación web, se consigue que todas las peticiones pasen por el Servlet creado que se encargará de redirigir a los diferentes manejadores las peticiones.

El funcionamiento del Servlet anteriormente comentado es el siguiente: en el *framework* Web MVC una vez iniciado el Servlet o *DispatcherServlet*, el sistema busca un fichero llamado "[servlet-name]-servlet.xml" en el directorio "WEB-INF" y crea los *beans* allí definidos. Cuando llega una petición y no encuentra ningún *bean* definido en el fichero pasa a buscarlo al aplicationContext raíz del proyecto. Este funcionamiento queda descrito en la Ilustración 21.

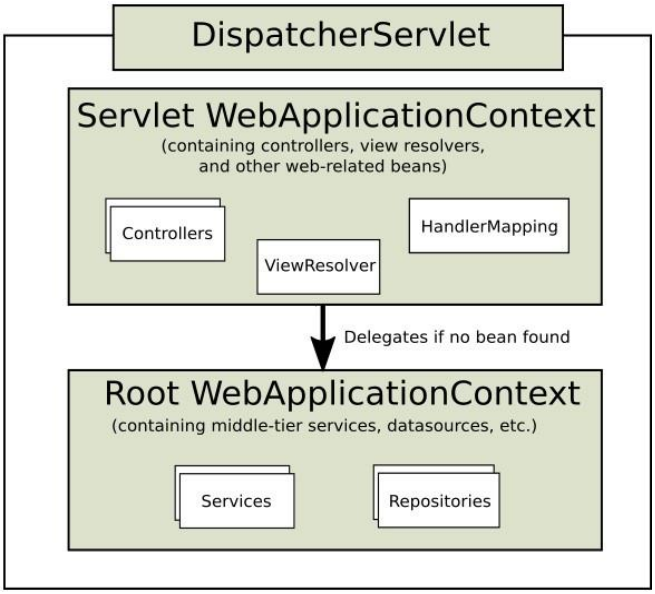



*Ilustración 21 Configuración de Spring para Web*

Una vez la estructura del paquete web ha sido configurada, se pasa a la creación o personalización de las respuestas. Es importante configurar la respuesta como deseamos, pero aún más configurar los mensajes de error para que envíen información al usuario, es decir, configurar unos mensajes de error descriptivos. Esto ayuda en caso de algún error del sistema la fácil localización y solución y en caso de error del usuario que este no derive a una incidencia técnica.

En este punto con Spring correctamente configurado, las dependencias añadidas y la base de datos llena de registros, es el momento de probar si todo lo realizado hasta el momento funciona. Para ello se ha creado un servicio web que recoge todas las provincias almacenadas en nuestra base de datos y las manda como respuesta a una petición realizada a través del método GET del protocolo HTTP.

Con este sencillo programa probamos que los *frameworks* tanto Spring como MyBatis funcionan correctamente, y en caso de no hacerlo, el hecho de haber hecho un programa de lógica sencilla nos ayuda a descartar los errores de configuración y encontrar los lógicos con mayor facilidad.

Si el programa responde correctamente a las peticiones enviadas a través de nuestro navegador implica que podemos comenzar a escribir la lógica de nuestro servicio de generación de fecha máxima de entrega sin preocuparnos de errores de configuración.

Una vez el código ha sido escrito, necesitamos probar su funcionamiento. Para ello mediante el *plug-in* instalado en nuestro entorno de desarrollo de Maven podemos compilar el código, empaquetarlo en un archivo de extensión EAR y desplegarlo en nuestro servidor para probarlo.

Para desplegar el archivo se puede hacer manualmente o automatizarlo mediante nuestro entorno de desarrollo. Al desplegarlo el servidor realiza una compilación previa y muestra la traza de la rutina ejecutada. En caso de dar error, este queda registrado y se puede consultar para su correcta solución. Si el despliegue es correcto ya podemos contactar con nuestro servicio para comenzar con las pruebas y la depuración del código.

## 5.3 Creación del proyecto front-end

Para creación de la interfaz del proyecto se ha seleccionado la tecnología REACT contenida en el entorno de ejecución multiplataforma conocido como Node.js. A lo largo de este apartado se mostrará y explicará el trabajo que se ha realizado para la creación de dicha interfaz.

Primero de todo es descargar Node.js desde su página oficial e instalar la versión 8.10 o posteriores. Después habrá que consultar si la versión del gestor de paquetes que viene por defecto en Node conocido como npm es la 5.6.0 o superior, en este caso se empleará la versión '6.12.0'.

Una vez instalado será necesario configurar las variables de entorno para que en la consola se puedan ejecutar comandos encabezados por 'npm' como 'npm --version'. Si el sistema permite este tipo de comandos, no hará falta configurar nada ya que el propio programa habrá añadido la ruta a las variables de entorno.

El segundo paso será instalar las librerías de REACT introduciendo en la consola 'npm install -g create-react-app'. Cuando haya terminado de descargar podremos ejecutar el comando 'npx create-react-app [nombre del proyecto]' y el sistema creará una aplicación en blanco con todas las configuraciones necesarias como se muestra en la Ilustración 22.

Para el desarrollo del código se ha utilizado el editor de código Visual Estudio Code que ofrece una interfaz intuitiva y permite diferenciar la sintaxis del código mediante colores. Esto ayuda a detectar errores y a una mejor organización del contenido.

Si en el editor abrimos la carpeta creada mediante la ejecución del comando 'npx create-react-app [nombre del proyecto]', se habrá generado en ella una estructura de directorios. Entre los cuales se pueden distinguir

los siguientes: el directorio node_modules donde residen todas las librerías necesarias para el desarrollo de aplicaciones; el archivo package.json donde se especificará las librerías externas que se emplearán durante el proyecto como la librería de componentes de Ant Design; el directorio public donde reside todo lo que contiene cualquier aplicación web dinámica con el fichero index.xml; y por último el directorio src donde se trabajará la mayoría del tiempo y donde reside el fichero *App.js* y todos los componentes que conforman la web.

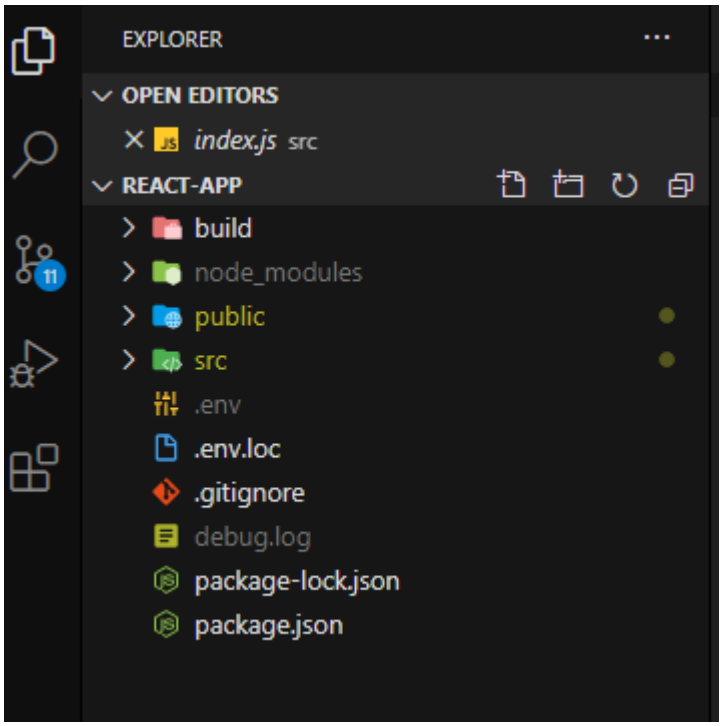



*Ilustración 22 Aplicación Vacía REACT*

Cuando la estructura de directorios ya ha sido creada, el próximo paso es crear la estructura lógica de la aplicación web. La tecnología seleccionada para este desarrollo es REACT que funciona mediante *JSX* que es la fusión de JavaScript y HTML. Resumiendo, esta tecnología permite almacenar código HTML en constantes de JavaScript que luego serán interpretadas y renderizadas por el navegador web. Un ejemplo de ello es el documento *index.html* mostrado en la Ilustración 23 que muestra como simplemente renderizando el componente <App/> en el elemento cuya id es root nuestra aplicación entera aparece en el navegador.

index.js

src > index.js

```
1  import 'react-app-polyfill/ie11';
2  import 'react-app-polyfill/stable';
3
4  import React from 'react';
5  import ReactDOM from 'react-dom';
6  import App from './App';
7
8  import './main.scss';
9
10 import * as serviceWorker from './serviceWorker';
11
12
13 ReactDOM.render(<App />, document.getElementById('root'));
14
15 // If you want your app to work offline and load faster, you can change
16 // unregister() to register() below. Note this comes with some pitfalls.
17 // Learn more about service workers: https://bit.ly/CRA-PWA
18 serviceWorker.unregister();
```

*Ilustración 23 Index.html*

Para el desarrollo de esta web se ha elegido la configuración de enrutamiento dinámico, es decir, nuestra aplicación no definirá todas las rutas desde el momento que se inicie, sino que se definirán a medida que se vayan renderizando. Se ha elegido así debido a la naturaleza de la aplicación web en la que se van a almacenar datos de los clientes y se requiere una interacción constante.

Por otro lado, como es necesaria la interacción de la aplicación con el servidor que alberga el programa de cálculo de fecha de entrega máxima, será necesario generar una manera de comunicarlos. Para ello se ha seleccionado la librería *Axios* de JavaScript que permite ejecutar llamadas en el navegador empleando el protocolo HTTP. Mediante esta librería se pueden realizar solicitudes y recibir respuestas fáciles de procesar. Dado que nuestro servidor está organizado mediante REST la funcionalidad que ofrece esta librería encaja a la perfección.

El siguiente paso es elegir los componentes que vamos a renderizar en nuestra aplicación web. De entre una gran variedad de librerías de componentes se ha seleccionada la conocida como *Ant Design*. Para ello será necesario modificar, como muestra la Ilustración 24, el archivo package.json donde figuran todas las librerías que emplea el proyecto añadiendo tanto esta como todas las comentadas hasta el momento.

```
{
  "name": "react-app",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "ant-design": "^1.0.0",
    "antd": "^3.26.18",
    "axios": "^0.19.2",
    "cpr": "^3.0.1",
    "node-sass": "^4.14.1",
    "react": "^16.13.1",
    "react-dom": "^16.13.1",
    "react-redux": "^7.2.0",
    "react-router-dom": "^5.1.0",
    "react-scripts": "^3.4.3",
    "redux": "^4.0.4",
    "swagger-express-middleware": "^2.0.3",
    "typescript": "^3.2.1"
  },
```

*Ilustración 24 package.json*

Una vez terminada la configuración del proyecto web solo faltará implementar las páginas con sus componentes asociados. En este caso se trata de 5 páginas con varios componentes en ellas. El caso de las páginas de mantenimiento contendrá dos componentes, un filtro y una tabla, y en el caso de las de alta contendrán un único filtro que actuará de modo de formulario.

Por último, cuando los componentes y sus páginas hayan sido implementadas se crearán variables JavaScript que contendrán el código HTML necesarios para que el navegador los renderice. Estas constantes serán añadidas en el componente <App/> que será el renderizado en primera instancia por el navegador. En Ilustración 25 se muestra cómo se añaden las variables de las páginas generadas a la variable <App/> que será la insertada en el index.html como se indicó en la ilustración 23.

```
return (
  <Router >
    <div className="App">
      <Layout style={{height:"100vh"}}>
        <Header name={messages.tituloAplicacion} />
        <MHorizontal modo="horizontal" tema="dark" items={navegacion} cambiarSubMenu={this.cambiarSubMenu} />
          <Layout style={{ marginLeft:'2%', marginRight:'2%'}}>
            <Sider className='sider'>
            {menuLateral}
            </Sider>
            <Content >
              {migasDePan}
              <Switch>
                <Route path='/mantenimientoRutas' component={MantenimientoRutasPage}/>
                <Route path='/mantenimientoTarifas' component= {mantenimientoTarifasPage}/>
                <Route path='/mantenimientoHorarios' component={MantenimientoHorarios}/>
                <Route path='/altaOficina' component = {AltaOficina}/>
                <Route path ='/altaTransporte' component = {AltaTransporte}/>
              </Switch>
            </Content>
          </Layout>
        <Footer/>
      </Layout>
    </div>
  </Router>

);
}
```

*Ilustración 25 Contenido de la variable <App/>*

# 6. Implantación

En el transcurso del siguiente apartado se van a comentar las acciones necesarias que se deben tomar para desplegar este servicio y hacerlo accesible desde cualquier punto.

Para comenzar será necesario un terminal con una dirección IP estática. Esto descarta el poder desplegarlo desde cualquier ordenador doméstico, ya que no disponemos de este tipo de IPs, por ello se comprarán instancias en un servidor de aplicaciones, un servidor web y una base de datos. Estos tres componentes que conforman la arquitectura, descrita en el apartado 4.1, son ofertados por Amazon.

La aplicación web puede ser desplegada a través de la plataforma de AWS Elastic Beanstalk[10] . Esta plataforma te permite de forma sencilla desplegar en internet cualquier servicio implementado en Java, .NET, Node.JS, Python, Ruby, Docker o Go. Dicha plataforma permite generar instancias de servidores virtuales en las cuales desplegar tu servicio y configurar la URL pública en la que desees ubicarlo.

Por otro lado, la parte web del programa, será desplegada en un servidor web Apache. Dicho servidor deberá contener nada más que la página web, pero con las variables de entorno actualizadas, es decir, la web deberá de realizar las llamadas a la URL del servidor de aplicaciones comentado en el párrafo anterior, que hará la función de backend y estará desplegado en otro dominio.

Por último, será necesario migrar la base de datos a una donde se tenga acceso desde el nuevo servidor de aplicaciones, para ello se puede utilizar la plataforma de bases de datos relacional de Amazon (RDS)[11] que permite instanciar una base de datos en la nube. Este servicio de Amazon oferta varios tipos de bases de datos como el de Oracle, el cual ha sido utilizado para el desarrollo del proyecto. Este tipo ofrece varias instancias con distintas especificaciones. Entre todas estas la instancia que más se adecua al sistema es la db.m5. Dicha instancia es de última generación y ofrece un equilibrio entre computación, memoria y recursos de red ideal para el proyecto implementado, y son una buena elección para la mayoría de las aplicaciones.

[10] https://aws.amazon.com/es/getting-started/hands-on/launch-an-app/

[11] https://docs.aws.amazon.com/es_es/AmazonRDS/latest/UserGuide/CHAP_Tutorials.WebServerDB.CreateDBInstance.html

# 7. Pruebas

Durante el transcurso del presente apartado se presentan las pruebas a las que será sometido el sistema y se documenta el resultado obtenido. Dichas pruebas se realizan con el fin de observar el comportamiento de la aplicación en un entorno lo más real posible.

Se realizan pruebas tanto para el sistema que tiene funciones directas con el usuario, como para el sistema que trabaja procesando las peticiones de generación de fechas de entrega. Para el primero se realizarán pruebas funcionales y para el segundo se realizarán tanto funcionales como de estrés. En los próximos subapartados se definirán dichas pruebas y se comentarán sus resultados.

## 7.1 Pruebas Backend

En este apartado se va a observar el comportamiento del servicio ante diferentes contextos que crearemos para estar seguros de que el servicio ofrece en todo momento la funcionalidad esperada.

Para realizar estas pruebas se enviarán diferentes peticiones al servicio sabiendo de antemano cual debe ser la respuesta y después se compararán. Además, a través de la aplicación JMetter se creará una situación de estrés en la que el servidor recibirá gran volumen de llamadas de diferentes focos para probar la robustez y observar su comportamiento.

Las primeras de las pruebas se pueden catalogar como pruebas de funcionalidad. Esta prueba consiste en predecir, a través de los datos almacenados, la respuesta de una petición en concreto. Mediante nuestro gestor de bases de datos vamos a consultar los datos que necesitamos para que la predicción sea lo más acertada posible. Una vez realizada la respuesta esperada se mandará la petición, a través de la aplicación PostMan o la interfaz de Swagger-ui, y se comprobará si la respuesta es la esperada.

### 7.1.1 Pruebas funcionales aplicación

Para la realización de este tipo de pruebas se han diseñado distintos contextos en los que se enviarán peticiones al servicio de generación de fechas de entrega máximas. Dichos contextos constarán de un día en específico el cual activará un flujo del servicio, cada contexto activará uno diferente, y una predicción de la respuesta basada en el funcionamiento esperado del servicio.

De este modo, si el servicio responde cumpliendo la predicción en todos los casos se comprobará que todos los diferentes caminos o flujos que toma el sistema para la generación de fechas de entrega máximas funcionan correctamente.

Una vez presentado el funcionamiento de las pruebas se comenzará con la primera prueba. Esta consta de una petición, mostrada en la Ilustración 26, generada a fecha 26/08/2020 a las 12:18:00. Se trata de un miércoles en horario de oficina, en este caso en el horario de la oficina número uno con destino la localidad de Badajoz con código postal "06001" y la tarifa seleccionada va a ser la estándar contratada por la empresa de dicha oficina.


```
{
    "codPostalDestino":"06001",
    "codOficina":"1",
    "codTarifaTransporte":"1",
    "fecAdmision":"2020-08-26T12:18:00"
}
```


*Ilustración 26 Prueba 1: Petición*

Dicha petición debe generar una respuesta con fecha de entrega máxima dos días después del lanzamiento de la petición, ya que la tarifa estándar contratada por la oficina uno tiene un número de días de trayecto igual a dos y no se encuentran festivos entre el transcurso de estos. En consecuencia, la fecha de entrega máxima debe de ser 28/08/2020 como se puede observar en la Ilustración 27.


```
{
    "success": true,
    "message": "Solicitud Correcta",
    "data": {
        "diasTarifas": 2,
        "diasFestivos": 0,
        "fechaRealAdmision": "2020-08-26T14:18:00.000+0200",
        "fechaEntregaMaxima": "2020-08-28T14:18:00.000+0200"
    }
}
```

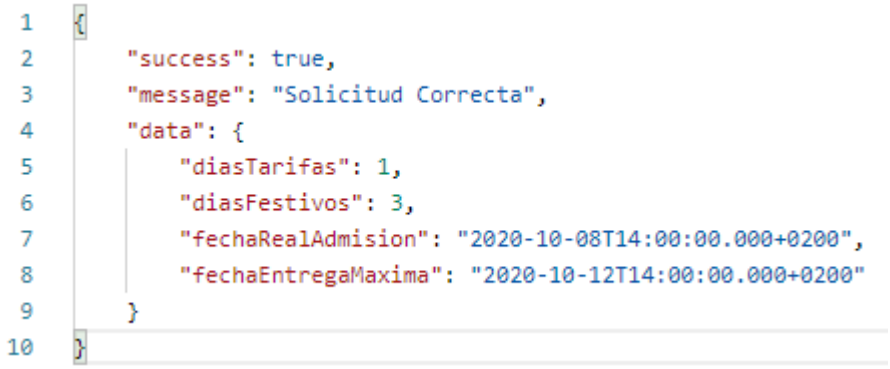

*Ilustración 27 Prueba 1: Respuesta*

La segunda prueba consta de una petición, mostrada en la Ilustración 28, enviada un jueves día 8/10/2020 a las 12:00:00, desde la oficina número uno hasta la localidad de Torrent, Valencia, con código postal 46900 y la tarifa seleccionada es la estándar contratada por la oficina.

```
{
    "codPostalDestino":"46900",
    "codOficina":"1",
    "codTarifaTransporte":"1",
    "fecAdmision":"2020-10-08T12:00:00"
}
```


*Ilustración 28 Prueba 2: Petición*

Según los días de trayecto definidos por la tarifa contratada por la oficina, el envío tardará un día en llegar a su destino. El inconveniente es que la fecha 9/10/2020 es festivo en la comunidad autónoma de Valencia y además es viernes. Por tanto, nos encontramos ante un festivo en destino que empalma con un finde semana lo que se traduce en tres días festivos. En conclusión, el sistema deberá generar una respuesta como la mostrada en la Ilustración 29 donde se suman cuatro días, uno de trayecto y 3 de festivos, a la fecha de admisión para generar la fecha de entrega máxima esperada, es decir, dicha fecha debe ser 12/10/2020.



```
{
    "success": true,
    "message": "Solicitud Correcta",
    "data": {
        "diasTarifas": 1,
        "diasFestivos": 3,
        "fechaRealAdmision": "2020-10-08T14:00:00.000+0200",
        "fechaEntregaMaxima": "2020-10-12T14:00:00.000+0200"
    }
}
```


*Ilustración 29 Prueba 2: Respuesta*

La tercera prueba consta de una petición, mostrada en la Ilustración 30, que se recibe un miércoles con fecha 22/04/2020 en la oficina con código seis, con destino Vizcaya y el pedido será enviado a través de la tarifa contratada. Esta fecha es festivo autonómico en la comunidad a la que pertenece la oficina, es decir, la petición se realiza mientras es festivo en el origen.


*Ilustración 30 Prueba 3: Petición*

El resultado esperado es una fecha real de admisión un día después de la fecha en la que se envía la petición, debido a que la oficina se encuentra cerrada a causa del festivo autonómico y procesará el envío en su siguiente día hábil. En este caso, no se informa del día festivo porque no transcurre durante el trayecto del envío, sino que se retrasa la admisión del pedido un día debido al cierre local de la oficina que lo gestiona. Por otro lado, la fecha de entrega generada en la respuesta, mostrada en la Ilustración 31, será de un día después de la fecha de admisión ya que los días por trayecto es uno.

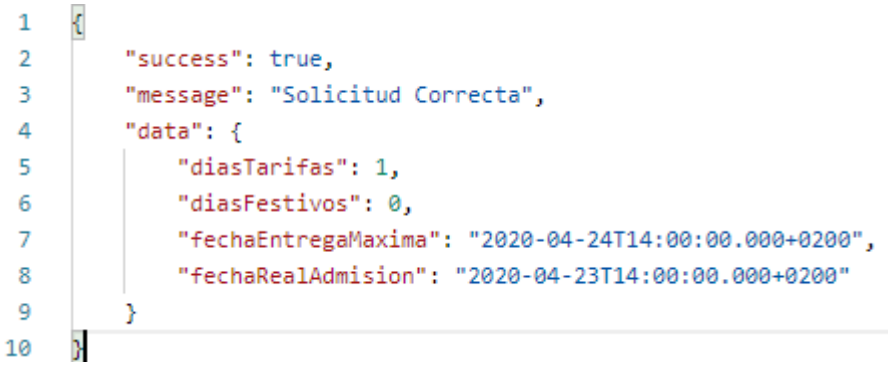



*Ilustración 31 Prueba 3: Respuesta*

La última de las pruebas funcionales consta de una petición como la mostrada en la Ilustración 32 que será enviada desde las Islas Baleares en concreto Palma de Mallorca con código postal 07001, hacia la oficina número uno, con la tarifa estándar contratada por la oficina y con la fecha 9/10/2020 a medio día.


*Ilustración 32 Prueba 4: Petición*

En esta petición los días de transporte entre las provincias asignados a la tarifa número uno es tres, entre los cuales va a transcurrir un festivo nacional, el día de Hispanidad, y un fin de semana. Por tanto, la fecha de entrega que se debe generar el sistema es seis días después de la fecha real de admisión (tres días festivos y tres días por tarifa) como se muestra en la Ilustración 33.


```
{
    "success": true,
    "message": "Solicitud Correcta",
    "data": {
        "diasTarifas": 3,
        "diasFestivos": 3,
        "fechaRealAdmision": "2020-10-09T14:00:00.000+0200",
        "fechaEntregaMaxima": "2020-10-15T14:00:00.000+0200"
    }
}
```


*Ilustración 33 Prueba 4: Respuesta*

### 7.1.2 Pruebas de estrés

Las siguientes pruebas sirven para tener una idea del comportamiento del servicio en situaciones específicas. Se van a simular situaciones de envío de un gran volumen de peticiones simultaneas al servidor para observar como éste las gestiona. Se tendrá en cuenta el tiempo y la cantidad de peticiones que el servidor es capaz de procesar.

Para ello, mediante el programa JMetter de Apache, se va a simular el envío de peticiones desde cuatro puntos diferentes de manera simultánea. Cada punto va a mandar tres peticiones diferentes, para que el flujo del programa no sea el mismo en todos los casos y se repetirá el envío de estas peticiones cada segundo.

Este contexto generará una situación de estrés que obligará al servidor a poner en funcionamiento su sistema de colas y a replicarse para aumentar la producción. Satisfaciendo así todas las peticiones con sus correspondientes respuestas.

Como se observa en la Ilustración 34 el plan de prueba está compuesto por cuatro grupos de hilos que ejecutarán tres peticiones cada grupo por segundo, es decir, se ejecutarán doce peticiones de cuatro fuentes diferentes cada segundo de manera ininterrumpida. Cada grupo de hilos contiene tres peticiones, una cabecera que define el cuerpo del mensaje y dos observadores que informarán sobre las peticiones. En el plan raíz también hay dos observadores que serán los que nos den la información global sobre el procesamiento de las peticiones lanzadas.

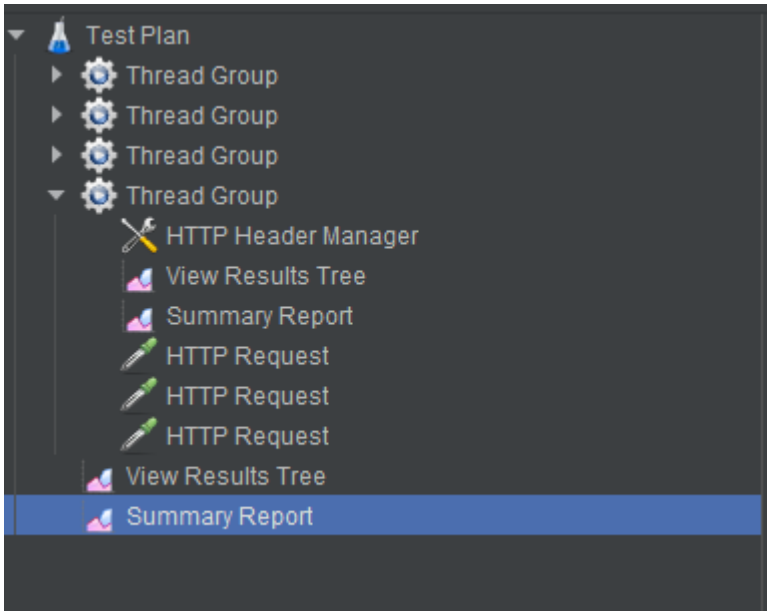



*Ilustración 34 Plan prueba de estrés.*

Los observadores nos darán información sobre: el nombre de cada tipo de petición en la columna Label, la cantidad de peticiones enviadas en la columna Samples, el tiempo que tarda el servidor en procesarlas en la columna Throghput , la desviación típica del procesado de las peticiones de cada tipo en la columna Std. Deviation y el tiempo medio que tarda en procesar las peticiones de cada tipo en la columna Average.

Los resultados obtenidos por la prueba de estrés mostrados en la Ilustración 35 indican que el sistema cumple con los requisitos especificados. Los resultados muestran que el servidor es capaz de comunicar seis peticiones y media por segundo por lo que si dividimos un segundo por el total de peticiones mostrado en el Throghput obtendremos los segundos que tarda en procesar una petición el servidor. En este caso obtenemos que cada petición se ha procesado en 0.1538 segundos período de tiempo que entra en el intervalo de 200 milisegundos definido por Google Developers como óptimo.

| Label | # Sa... | Aver... | Min | Max | Std. D... | ... | Throughput | Received ... | Sent K... |
|---|---|---|---|---|---|---|---|---|---|
| HTTP Req... | 109 | 647 | 329 | 1147 | 142.40 | ... | 2.2/sec | 0.83 | 0.72 |
| HTTP Req... | 107 | 472 | 130 | 711 | 100.85 | ... | 2.2/sec | 0.78 | 0.72 |
| HTTP Req... | 106 | 711 | 285 | 1056 | 147.59 | ... | 2.2/sec | 0.78 | 0.72 |
| TOTAL | 322 | 610 | 130 | 1147 | 166.04 | ... | 6.5/sec | 2.37 | 2.13 |

*Ilustración 35 Resultado prueba de estrés*

Como se ha conseguido superar la prueba de estrés sin problemas se va a llevar al límite el servicio. Ahora se va a ejecutar el plan de pruebas con un volumen de peticiones sesenta veces mayor que el anterior. Es decir, en vez de cuatro nodos enviando tres peticiones por segundo vamos a replicar esos nodos veinte veces cada uno. En este caso, se simulará un contexto en el que el servicio recibirá ciento ochenta peticiones por segundo. Con este contexto queremos llevar al límite nuestro servicio para ver cómo se comporta y así poder analizar los resultados mostrados en la Ilustración 36.

| Label | # Samples | Average ↑ | Min | Max | Std. Dev. | ... | Throughput | Recei... | Sen... | Avg.... |
|---|---|---|---|---|---|---|---|---|---|---|
| HTTP Request2 | 125 | 9833 | 7020 | 13521 | 1095.18 | ... | 2.4/sec | 0.81 | 0.80 | 343.0 |
| HTTP Request1 | 160 | 13049 | 8410 | 18511 | 2101.51 | ... | 2.8/sec | 0.93 | 0.92 | 343.0 |
| HTTP Request3 | 80 | 13598 | 9586 | 16766 | 1588.23 | ... | 3.1/sec | 1.03 | 1.02 | 343.0 |
| TOTAL | 365 | 12068 | 7020 | 18511 | 2354.76 | ... | 6.1/sec | 2.03 | 2.00 | 343.0 |

*Ilustración 36 Resultado prueba de estrés x60*

Durante la ejecución del nuevo plan de pruebas se observa que, al no poder procesar todas las peticiones que llegan, el servidor las encola. Se observa que el volumen de peticiones que responde por segundo es un poco menor al caso anterior. Esto puede ser debido al retraso en el comienzo del procesamiento de las peticiones causado por la aglomeración de estas desde el primer momento. Esto conlleva a un tiempo mínimo de procesamiento de la respuesta mayor y, en consecuencia, la media aumenta. En este contexto el servidor es capaz de procesar una petición cada 0.1639 periodo de tiempo también incluido en el intervalo óptimo anteriormente comentado.

Los nuevos resultados nos indican que el servidor todavía responde a un gran volumen de peticiones por segundo suficiente para la demanda simulada en el plan de pruebas. Esto indica que el sistema responde sin problemas incluso en un contexto simulado que difícilmente llegará a ocurrir y lo que muestra es que aún con este gran volumen de peticiones el servicio no se cae y sigue ofreciendo funcionalidad, es decir, el servicio cumple con el requisito de disponibilidad, robustez y eficiencia.

## 7.2 Pruebas de funcionalidad de la aplicación web.

En este apartado se van a implementar las pruebas funcionales para la parte que funciona de *front-end*. En ellas se van a realizar acciones en el sistema esperando una respuesta anteriormente definida, es decir, se partirá de un estado inicial y de un estado final previamente definido, y luego, se someterá al

sistema a una correlación de acciones que deben terminar como había sido previsto.

Dichas pruebas se van a realizar por pantallas, es decir, se definirá un caso de uso para cada pantalla que conforma la web. Una vez definidos, se mostrarán mediante imágenes como se desarrolla la interacción con web siguiendo dicho plan.

### 7.2.1 Prueba Funcional: Mantenimiento Rutas

Para el caso de pruebas relacionado con la página de Mantenimiento de Rutas se va a definir un usuario el cual desea consultar el estado actual de la tarifa estándar de la empresa de transporte Coyote, eliminar un registro y limpiar su búsqueda.

Para ello debe abrir el menú horizontal de Transporte y hacer clic sobre el menú vertical en la opción Mantenimiento Rutas. El sistema mostrará un filtro con las indicaciones de obligatoriedad para realizar la consulta. Hasta que estas indicaciones no se cumplan el sistema no permitirá hacer clic sobre el botón de Buscar.

El estado inicial de dicha página, mostrado Ilustración 37 carga las empresas de transporte y provincias dadas de alta en la base de datos y las muestra en sus respectivos combos. Además, bloquea el siguiente combo, el referente a las tarifas, porque hasta que no se seleccione una empresa de transporte el sistema no puede cargar las tarifas asociadas a esta. Esto es debido a que el filtro carga su información de manera dinámica para facilitar su cumplimentación.

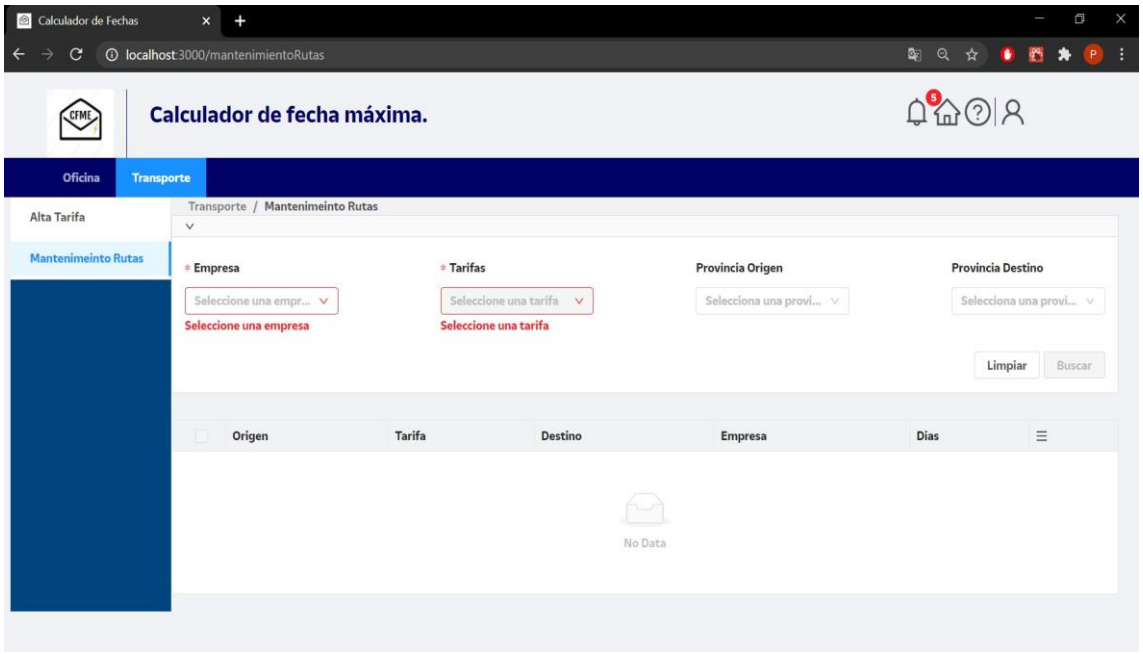



*Ilustración 37 Mantenimiento Rutas: Estado Inicial*

Una vez ha sido seleccionada la empresa de transporte el sistema habilita el combo de las tarifas y carga las tarifas asociadas a dicha empresa almacenadas en la base de datos como se muestra Ilustración 38. Por otro lado, el botón de Buscar sigue deshabilitado dado que el sistema requiere que los combos de empresa de transporte y tarifas sean cumplimentados.

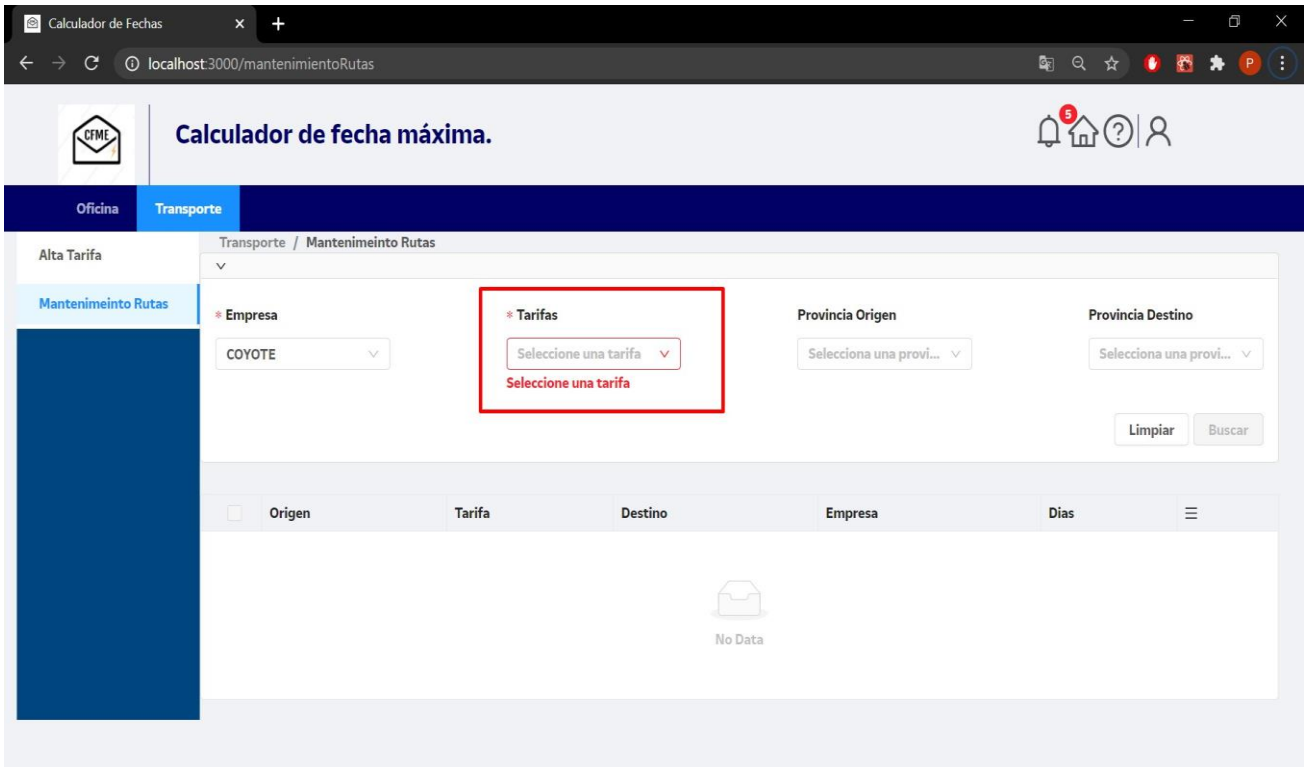



*Ilustración 38 Mantenimiento Rutas: primer paso*

Una vez ha sido seleccionada la empresa de transporte y la tarifa asociada, el sistema habilita el botón de Buscar como se muestra en la Ilustración 39. Permitiendo al usuario seguir cumplimentando el filtro para realizar una búsqueda más concreta, filtrando por las provincias de origen y destino, o realizar la búsqueda de la tarifa incluyendo todas las provincias.

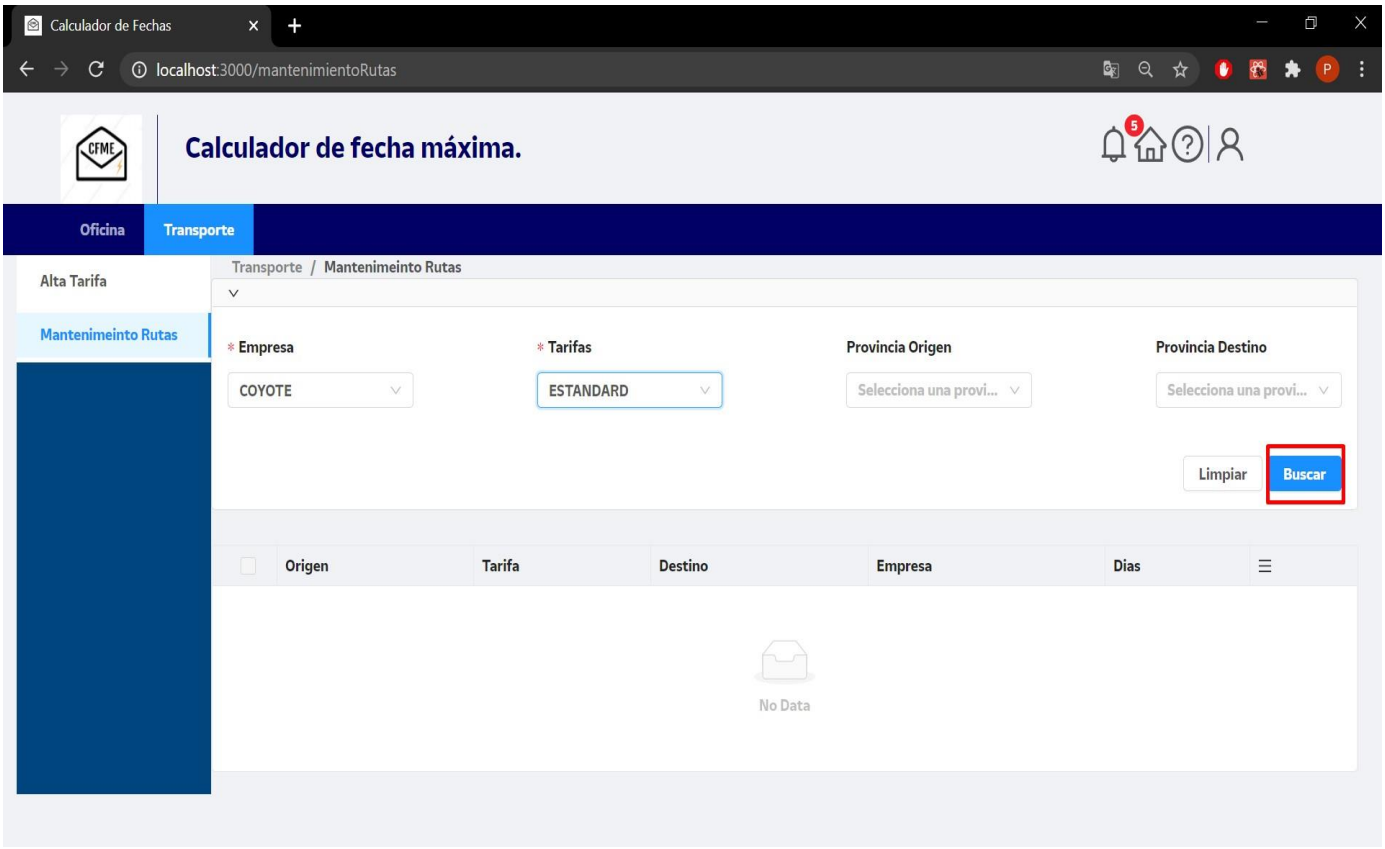



*Ilustración 39 Mantenimiento Rutas: segundo paso*

En este caso de uso el usuario pretende consultar una tarifa asociada a una empresa de transporte por lo que no rellena los campos de provincias y hace clic sobre el botón de Buscar. El sistema envía una petición al servicio de web asociado a dicha funcionalidad y responde con todos los registros asociados a la tarifa seleccionada. La aplicación web los recoge y los muestra en la tabla inferior como se muestra en la Ilustración 40. Por cada registro el sistema permite eliminar el registro deseado o limpiar la búsqueda dejando la página en su estado inicial.

El usuario desea eliminar el registro y hace clic sobre la opción asociada al registro de eliminar. El sistema muestra un bocadillo de confirmación y si es aceptado el registro será dado de baja en la base de datos, se le comunicaría al usuario el resultado de la operación y la tabla se volvería a cargar sin mostrar el registro eliminado como se muestra en la Ilustración 42 , en cambio, si la confirmación es cancelada el sistema vuelve al estado actual. En este caso como el usuario quiere dar de baja un registro acepta la confirmación mostrada en la Ilustración 41.

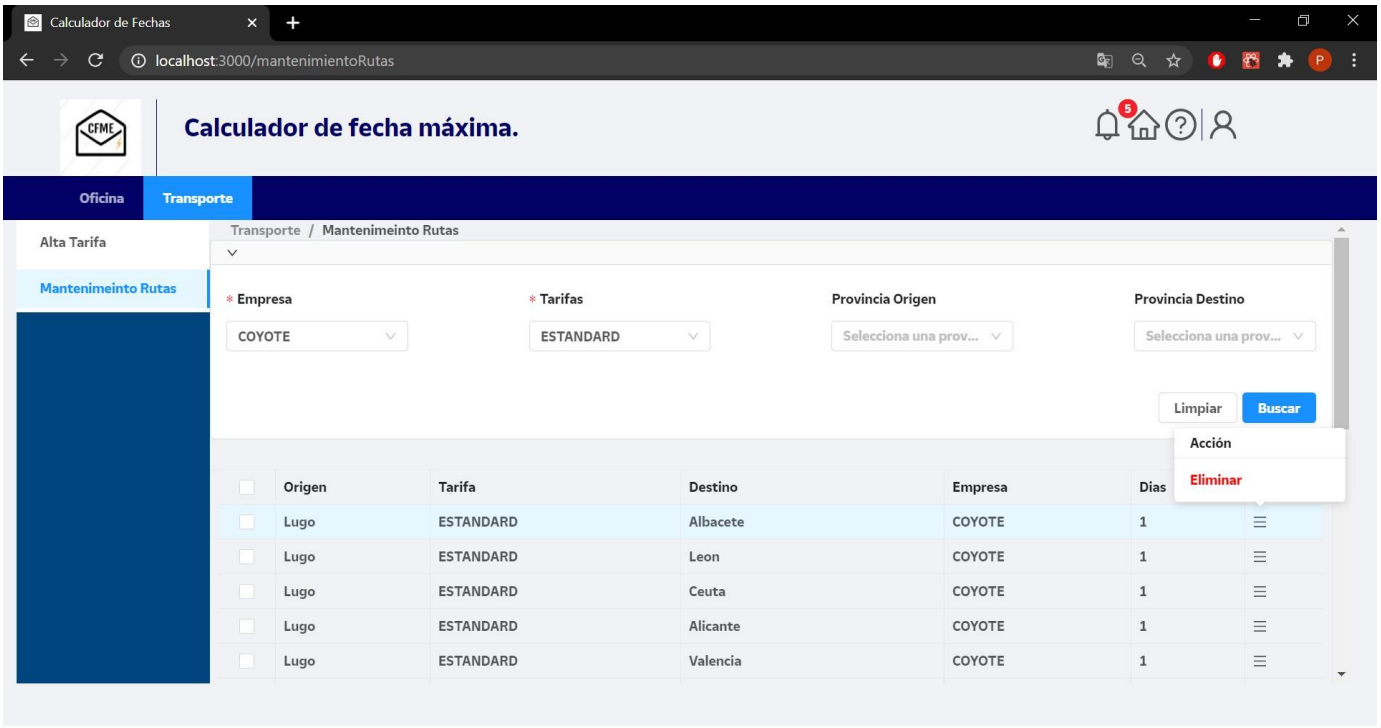



*Ilustración 40 Mantenimiento Rutas: tercer paso*

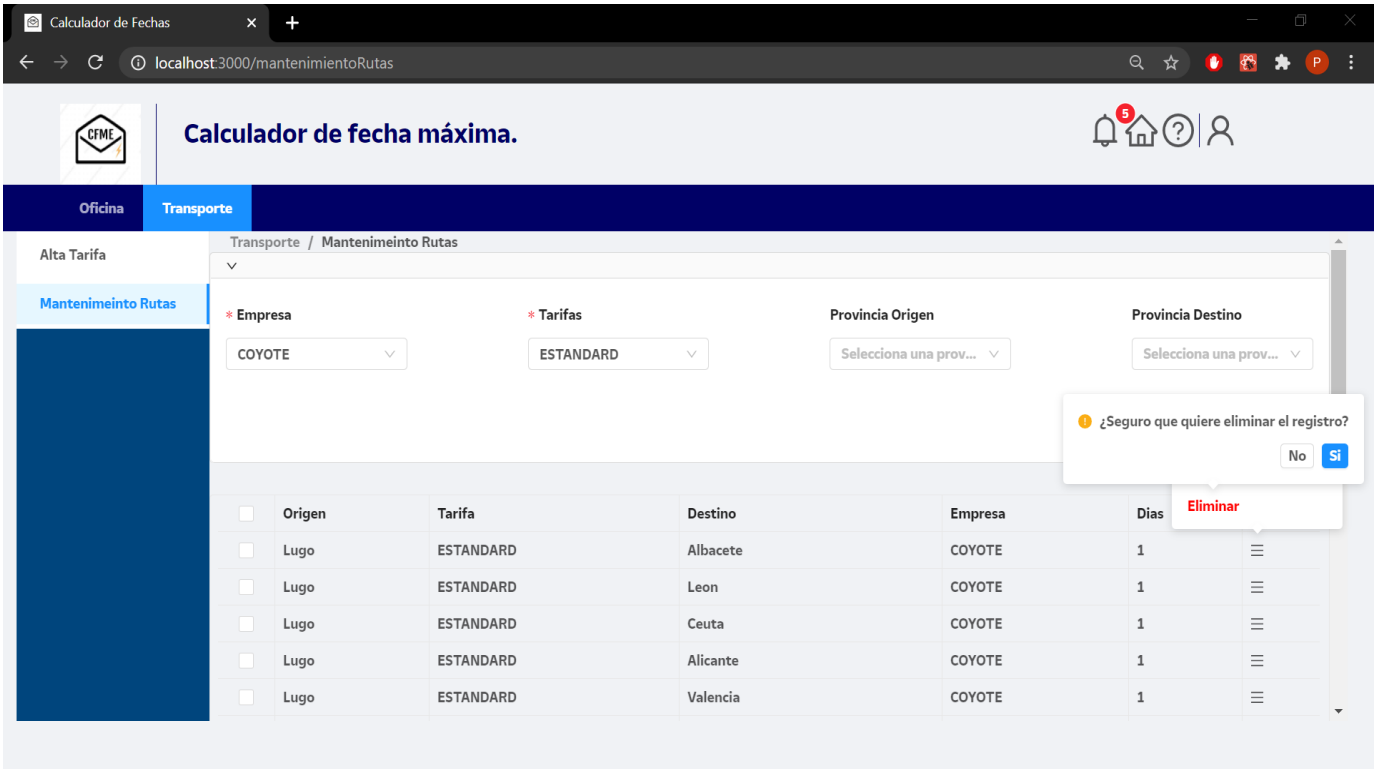



*Ilustración 41 Mantenimiento Rutas: cuarto Paso*

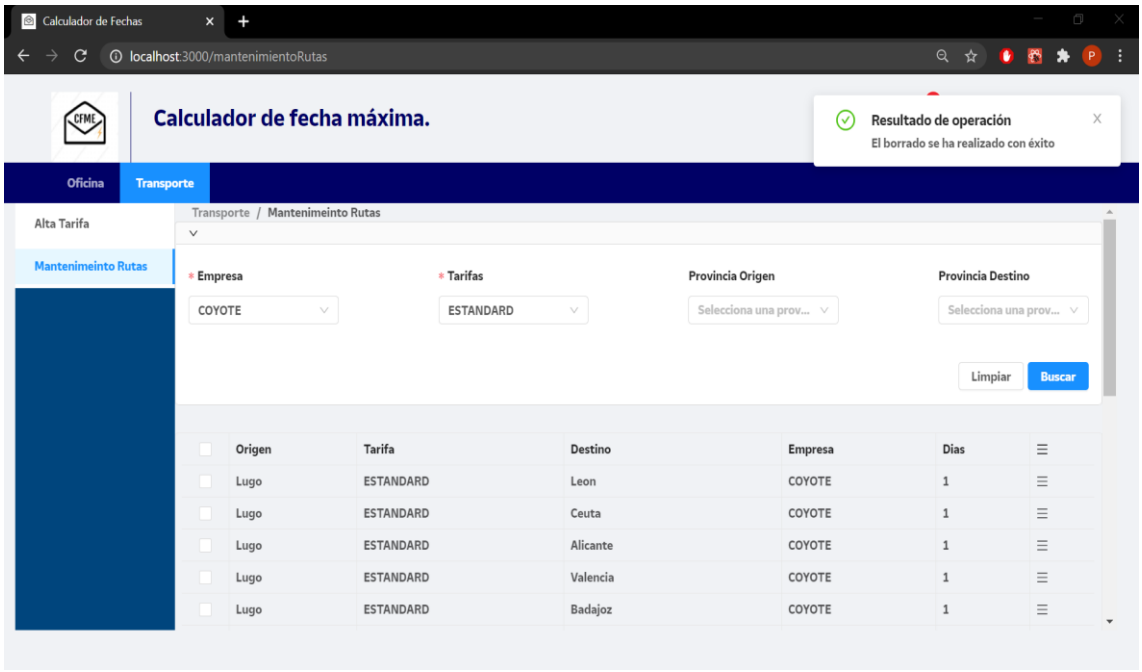



*Ilustración 42 Mantenimiento Rutas: quinto paso*

Una vez realizada la acción de búsqueda la tarifa asociada a la empresa y eliminado el registro deseado el usuario desea limpiar la página para que esta vuelva a su estado inicial. Haciendo clic sobre el botón limpiar el sistema oculta la tabla, limpia los combos rellenados del filtro, bloquea el botón de Buscar y vuelve a mostrar las indicaciones de obligatoriedad para realizar la búsqueda quedando la pantalla como se muestra en la Ilustración 43.

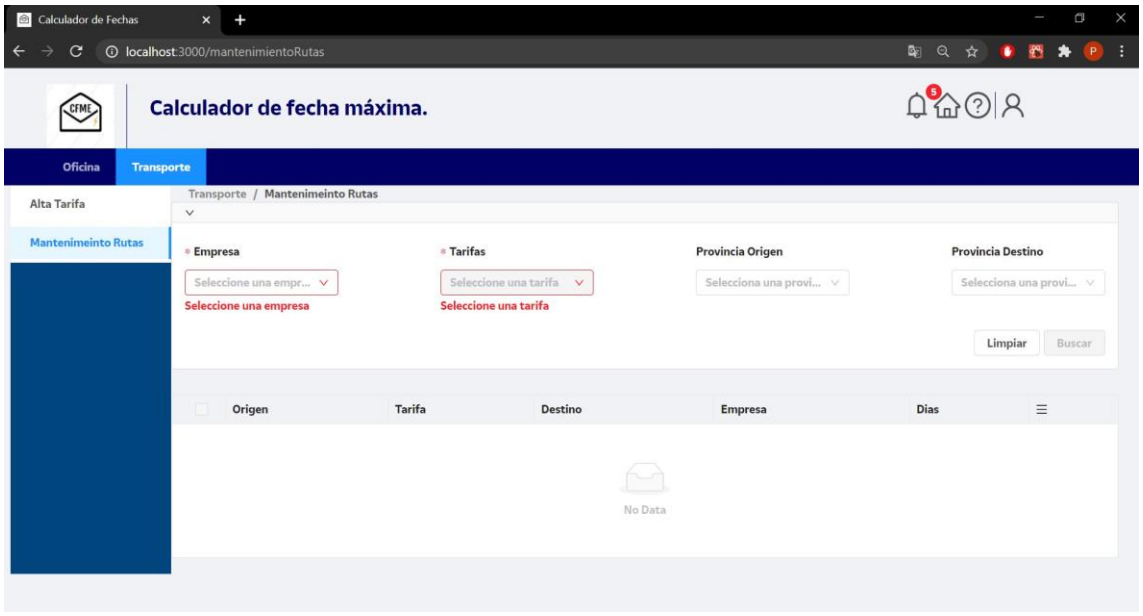



*Ilustración 43 Mantenimiento Rutas: sexto Paso*

### 7.2.2 Prueba Funcional: Mantenimiento Horarios

Para el caso de pruebas relacionado con la página de mantenimiento de horarios se va a definir un usuario el cual desea consultar el estado actual de las oficinas pertenecientes a una empresa.

Para ello debe abrir el menú horizontal de Oficina y hacer clic sobre el menú vertical en la opción Mantenimiento Horarios. El sistema mostrará un filtro con las indicaciones de obligatoriedad para realizar la consulta. Hasta que estas indicaciones no se cumplan el sistema no permitirá hacer clic sobre el botón de buscar.

El estado inicial de dicha página, mostrado en la Ilustración 44, carga las empresas cliente dadas de alta en la base de datos y las muestra en su combo. Además, bloquea el siguiente combo, el referente a las oficinas, porque hasta que no se seleccione una empresa el sistema no puede cargar las oficinas asociadas a esta. Esto es debido a que el filtro carga su información de manera dinámica para facilitar su cumplimentación.

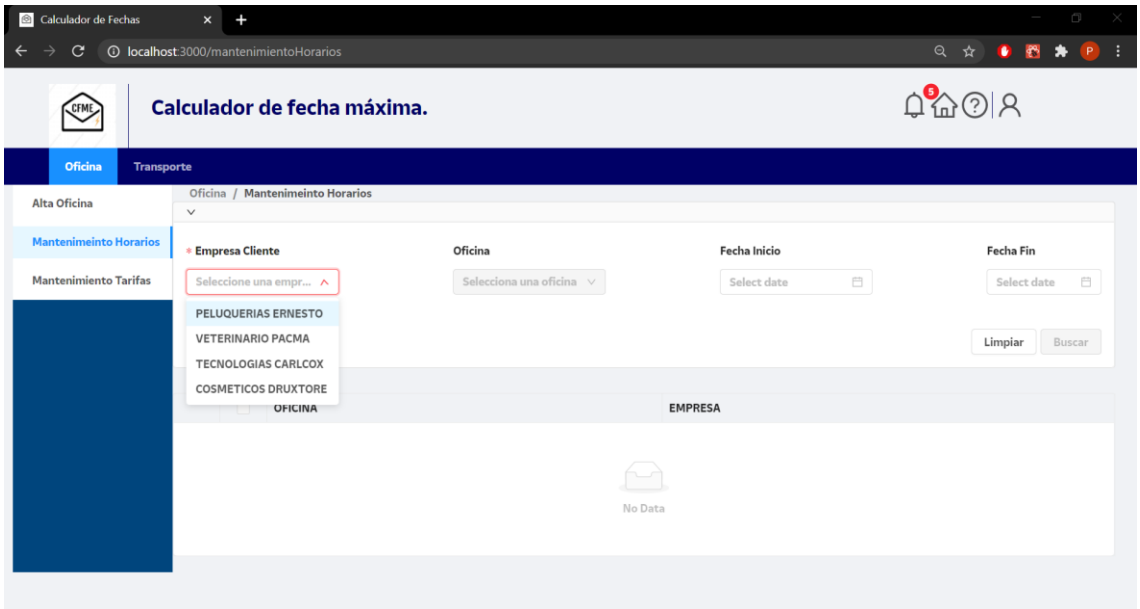



*Ilustración 44 Mantenimiento Horarios: estado Inicial*

Una vez ha sido seleccionada la empresa cliente el sistema habilita el combo de las tarifas y carga las descripciones de las oficinas, asociadas a dicha empresa, almacenadas en la base de datos como se muestra en la Ilustración45. Por otro lado, el botón de Buscar se habilita dado que el sistema requiere nada más que el combo de Empresa Cliente sea rellenado.

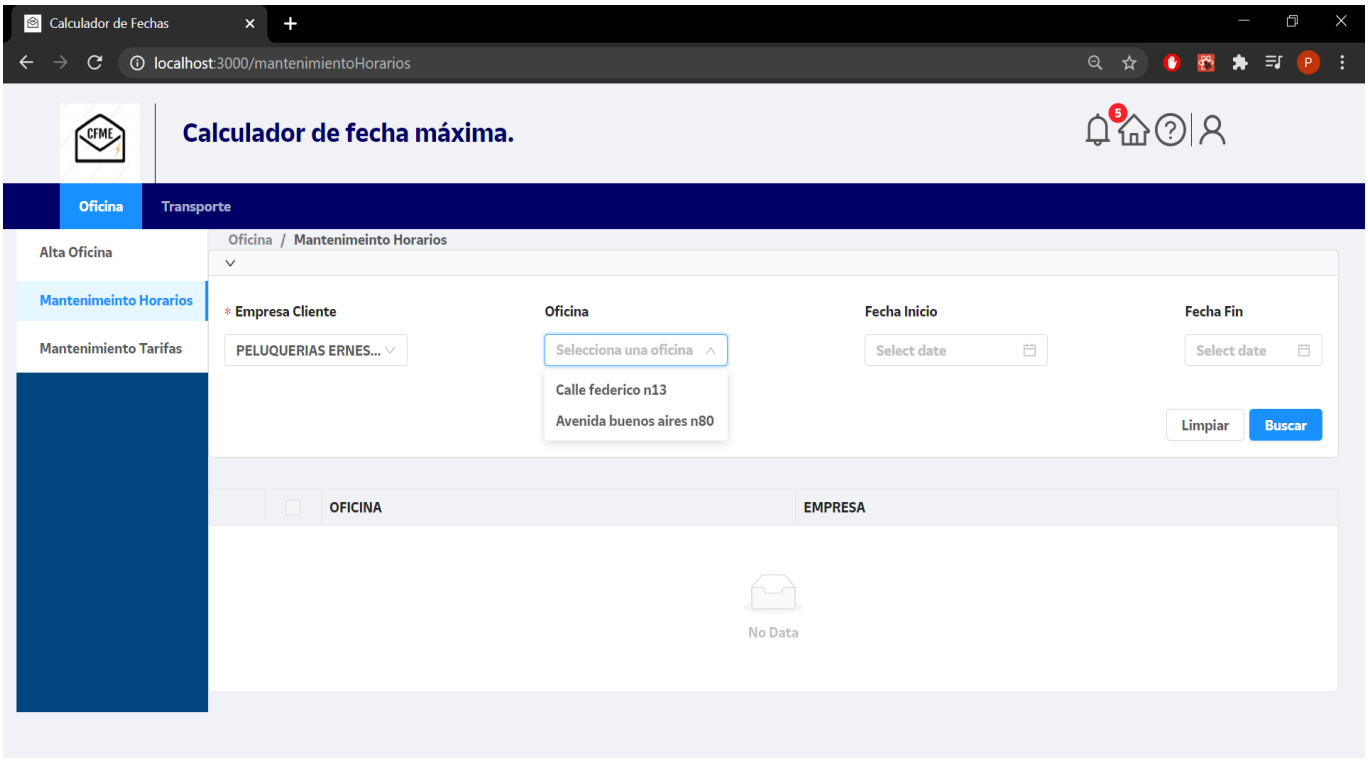



*Ilustración 45 Mantenimiento Horarios: primer paso*

En este punto el usuario puede elegir si seguir introduciendo datos para filtrar más los resultados o bien buscar todas las oficinas, asociadas a la empresa cliente seleccionada, con todos sus horarios de trabajo. En este caso, el usuario selecciona opción de buscar todas las oficinas asociadas con sus respectivos horarios por lo que hace clic en el botón de buscar y el sistema genera una petición al servidor *backend*. Este recibe la petición y genera una respuesta acorde la cual es recibida por la aplicación web. La aplicación muestra los datos de la búsqueda en la tabla de abajo como se observa en la Ilustración 46.

En dicha parte de abajo , se puede observar los registros desplegables de las distintas oficinas asicadas a la empresa. Si se despliegan dichos registros se muestran los horarios asociados a cada una de las oficinas.

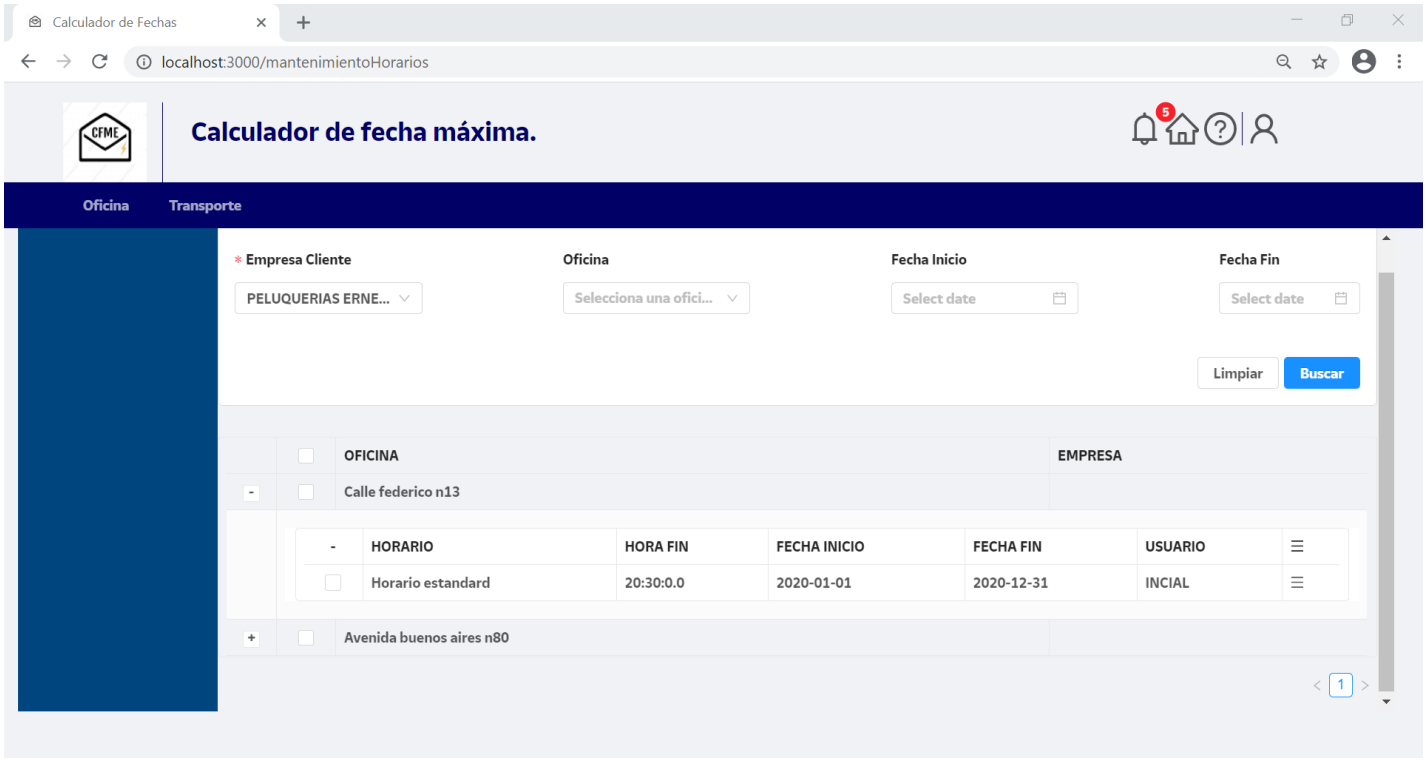



*Ilustración 46 Mantenimiento Horarios: tercer Paso*

### 7.2.3 Prueba funcional: Mantenimiento Tarifas

Para el caso de pruebas relacionado con la página de mantenimiento de tarifas se va a definir un usuario el cual desea consultar el estado actual de las tarifas asociadas a una oficina perteneciente a una empresa cliente y darse de baja de la tarifa de transporte.

El primer paso es abrir el menú horizontal de Oficina y hacer clic sobre el menú vertical en la opción Mantenimiento Tarifas. El sistema mostrará un filtro con las indicaciones de obligatoriedad para realizar la consulta. Hasta que estas indicaciones no se cumplan el sistema no permitirá hacer clic sobre el botón de Buscar.

La página de Mantenimiento de Tarifas, mostrada en la Ilustración 47, consta de un filtro el cual se rellena dinámicamente dependiendo de los datos que sean introducidos en los combos. El primer combo es el único obligatorio y es el relacionado con las empresas clientes, el segundo es el de oficinas que muestra las oficinas asociadas a la empresa cliente y el último es el relativo a las tarifas asociadas a la oficina seleccionada.

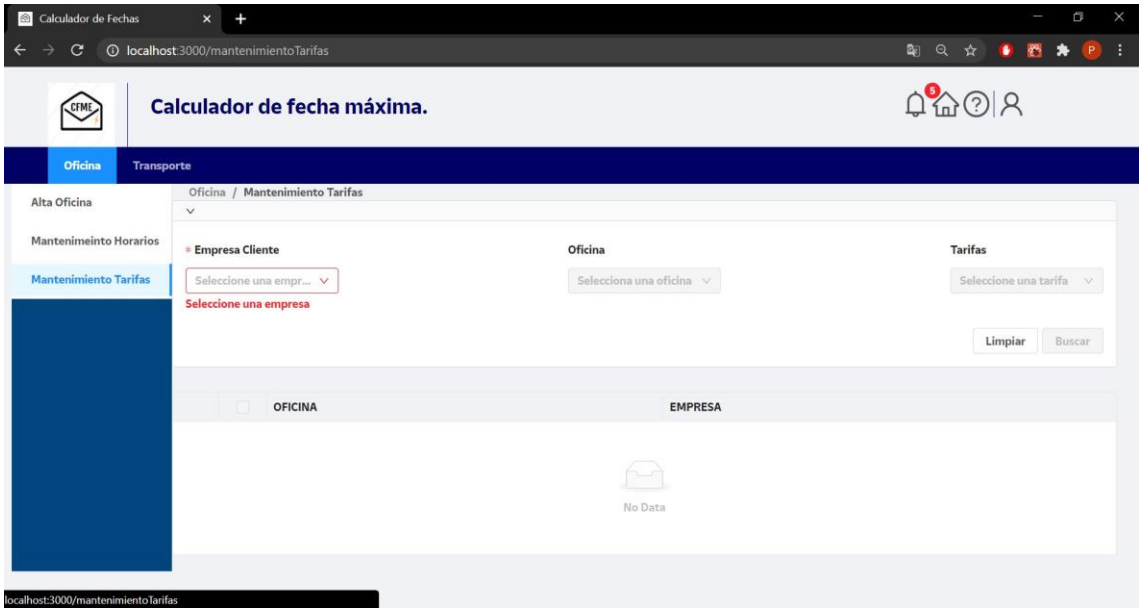



*Ilustración 47 Mantenimiento Tarifas: estado inicial*

El usuario selecciona su empresa y la oficina a la cual desea dar de baja la tarifa de transporte. El sistema va desbloqueando los combos a medida que el usuario introduce los datos como se puede observar en la Ilustración 48. Cuando el usuario tiene los datos introducidos, hace clic sobre el botón de Buscar.

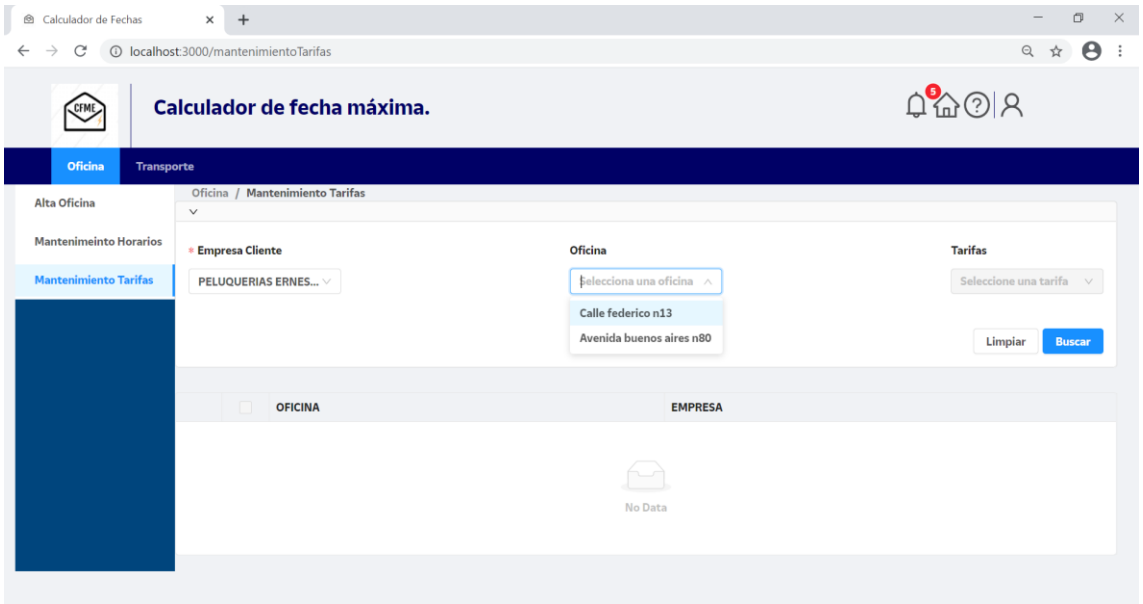



*Ilustración 48 Mantenimiento Tarifas: primer paso*

Al hacer clic sobre el botón de buscar el sistema genera una petición con los datos introducidos en los combos y la envía al servidor backend. Este genera una respuesta acorde a la petición y se la comunica a la aplicación web la cual muestra una tabla con los datos recibidos del servidor backend como se muestra en la Ilustración 49.

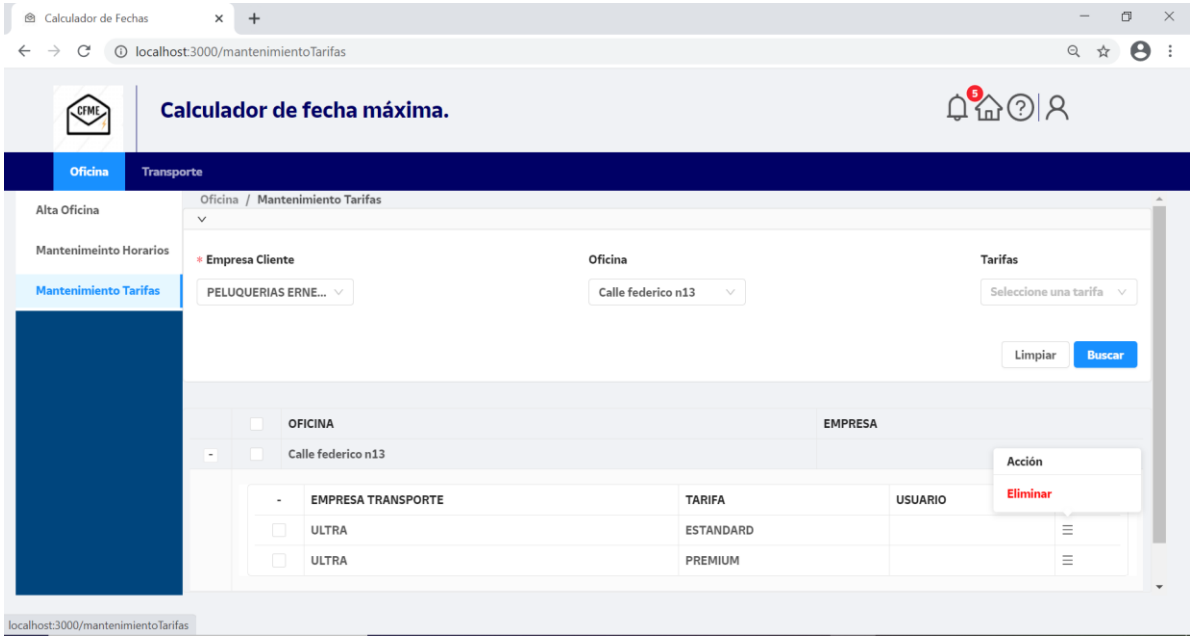



*Ilustración 49 Mantenimiento Tarifas: segundo paso*

El usuario hace clic sobre el registro de la tarifa asociada a la oficina que desea eliminar. El sistema muestra un bocadillo de confirmación, como el mostrado en la Ilustración 50, el cual si es aceptado el sistema eliminará el registro deseado y si es rechazado se mantendrá inalterado. En este caso el usuario acepta la confirmación y el sistema informa al usuario de que la eliminación del registro ha sido exitosa como se muestra en la Ilustración 51.

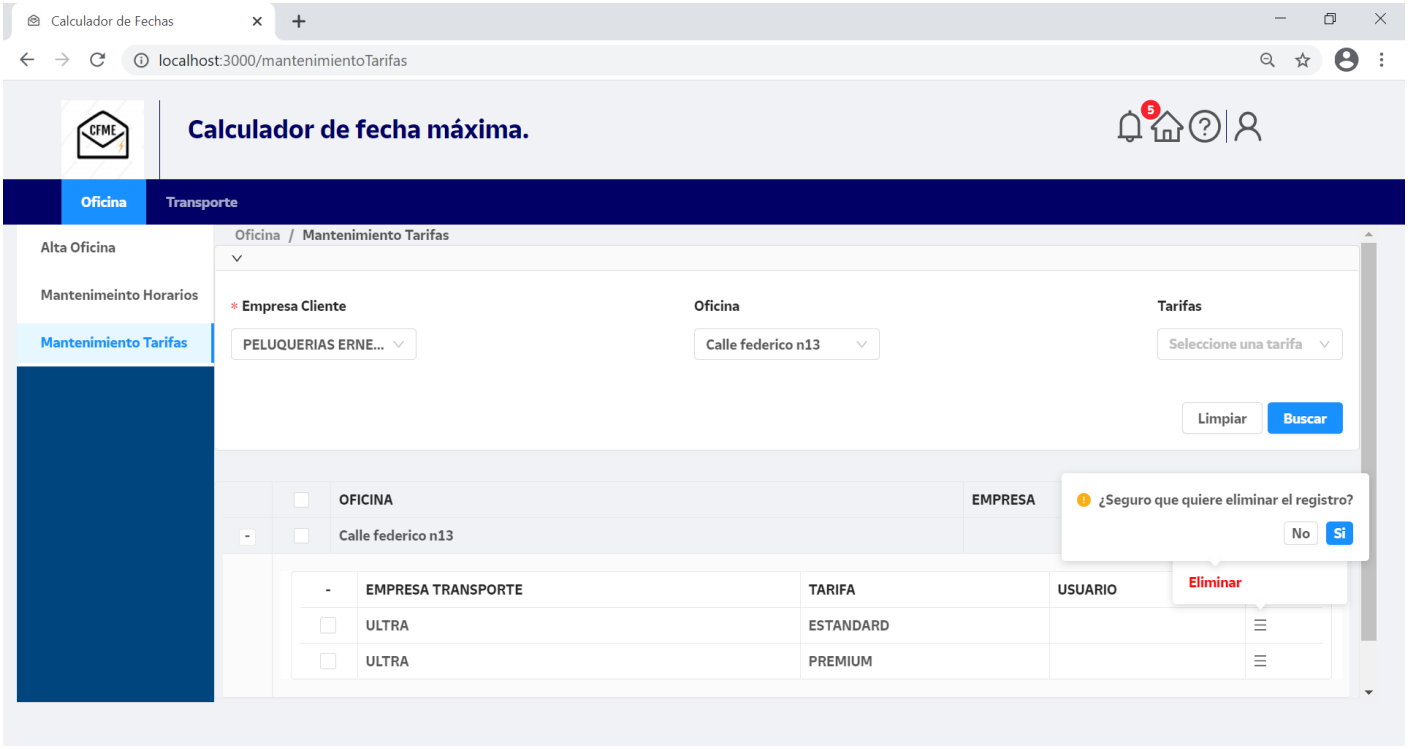



*Ilustración 50 Mantenimiento Tarifas: tercer paso*

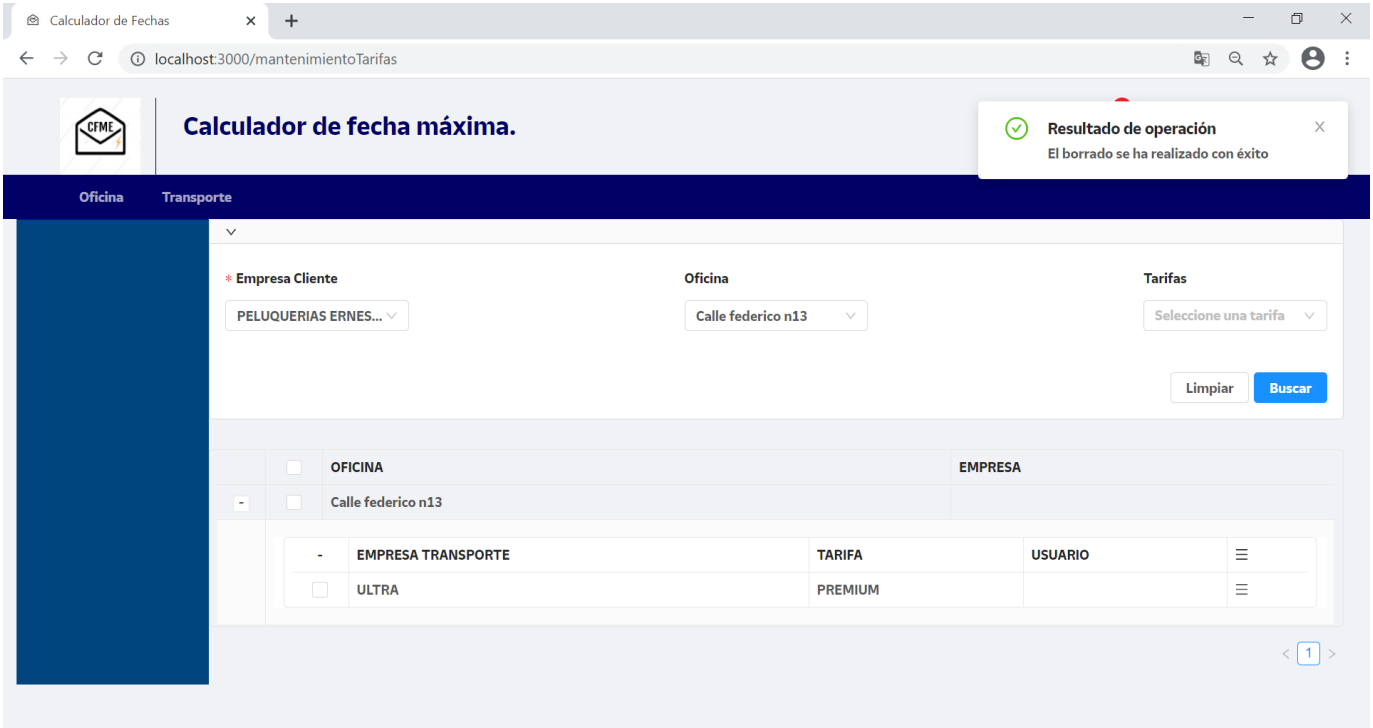



*Ilustración 51 Mantenimiento Tarifas: cuarto paso*

### 7.2.4 Prueba Funcional: Alta Tarifa

Para el caso de pruebas relacionado con la página Alta Tarifas se va a definir un usuario el cual desea dar de alta una nueva tarifa asociada a una empresa de transporte que no existe en la base de datos, es decir, quiere dar de alta una nueva empresa de transporte en el sistema y asociarle una nueva tarifa con diferentes rutas de entrega.

El primer paso es abrir el menú horizontal de Transporte y hacer clic sobre el menú vertical en la opción Alta Tarifa. El sistema mostrará un filtro con las indicaciones de obligatoriedad para realizar el alta. Hasta que estas indicaciones no se cumplan el sistema no permitirá hacer clic sobre el botón de Aceptar.

La página de Alta Tarifa, mostrada en la Ilustración52, consta de un formulario con los datos necesarios para dar de alta una nueva tarifa en el sistema. La funcionalidad de esta página es muy amplia ya que permite dar de alta una nueva empresa con una nueva tarifa y nuevos registros en esta. También permite el alta de una nueva tarifa de una empresa de transporte existente con nuevos registros de rutas asociados. Y, por último, permite dar de alta nuevos registros de rutas a tarifas existentes.

Al igual que en el resto de las páginas los combos son precargados con los datos almacenados en la base de datos. En este caso, según el caso de pruebas definido, el usuario quiere dar de alta su nueva empresa de transporte con una nueva tarifa asociada a esta. Entonces el usuario hace clic sobre el *checkbox* Nueva Empresa. El sistema cambia la estructura de la página cambiando los componentes *select* de Empresa Transporte y Tarifa por unos *inputs*. Además, detecta que al tratarse de una nueva empresa la tarifa será nueva obligatoriamente y marca el *checkbox* de Nueva Tarifa automáticamente.

El usuario puede introducir en los nuevos campos seleccionados los nombres de la empresa de transporte y la tarifa que quiere dar de alta como se muestra en la Ilustración 53.

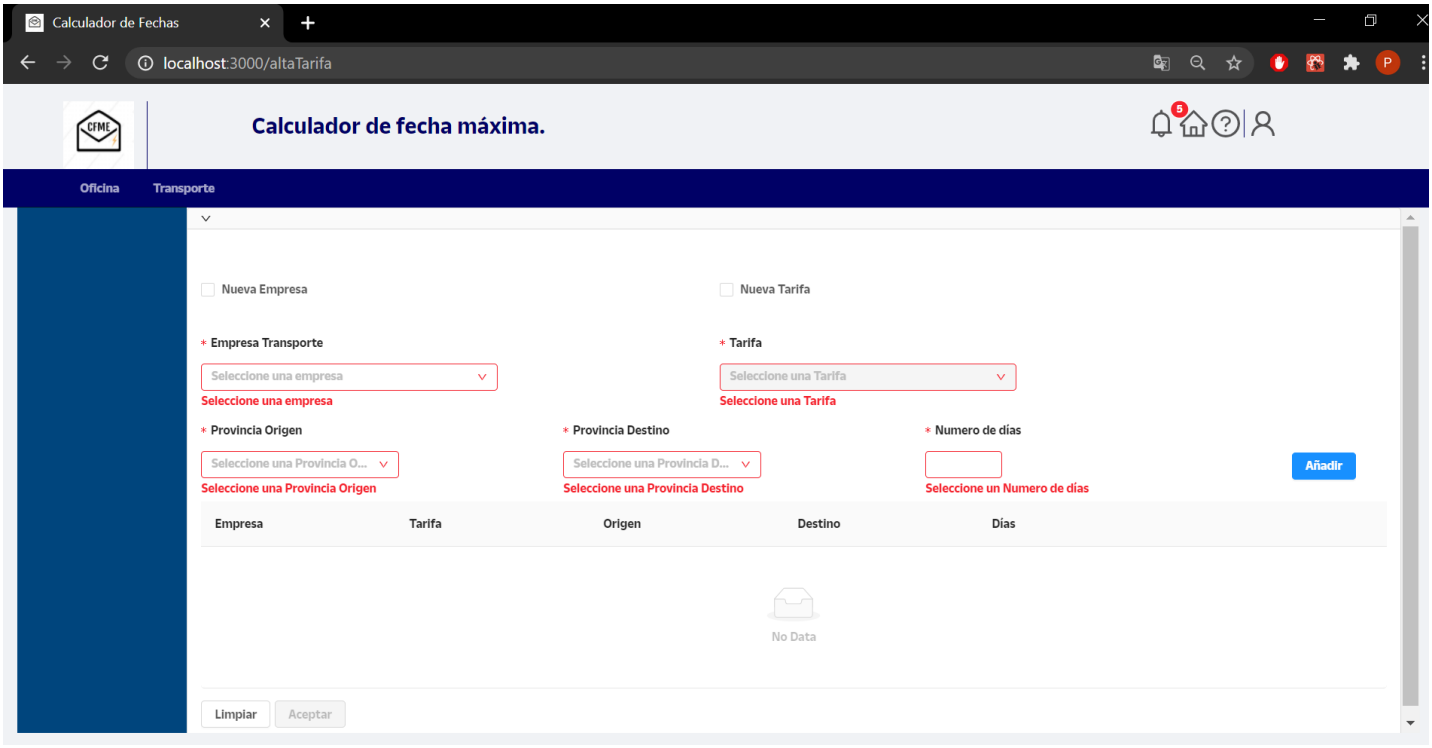



*Ilustración 52 Alta Tarifa: estado inicial*

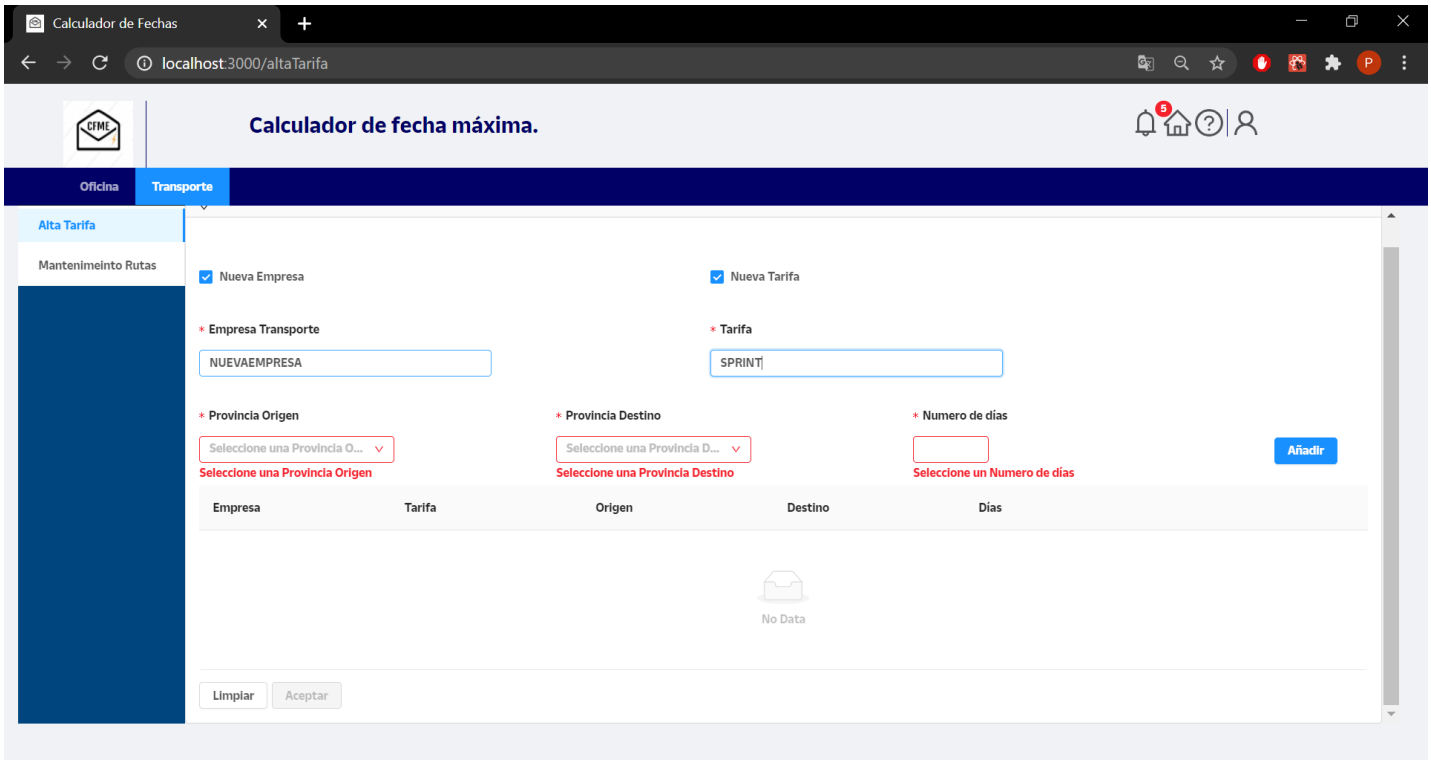



*Ilustración 53 Alta Tarifa: primer paso*

Una vez seleccionados los nombres de la empresa y la tarifa a dar de alta, solo faltan rellenar los registros de las rutas. En el formulario mostrado en la Ilustración 54 es necesario añadir la provincias, origen y destino y, el número de días que tarda la empresa en desplazarse entre estas.

Cuando los datos son rellenados en los combos el usuario hace clic en el botón Añadir limpiado el contenido de los combos y añadiendo un registro en la tabla que el usuario puede eliminar si no está conforme con los datos introducidos.

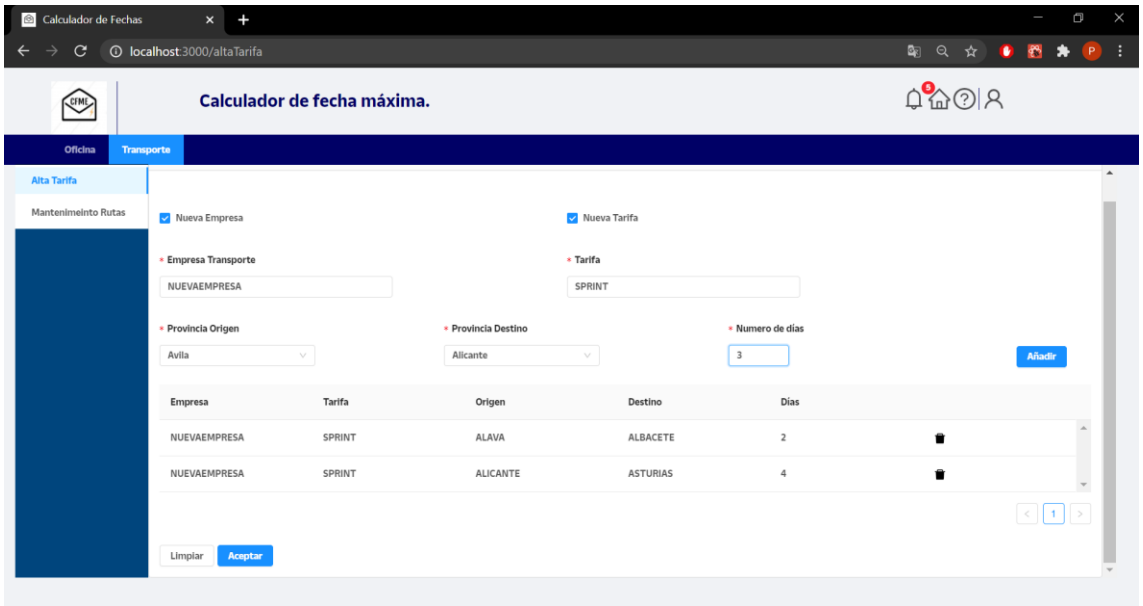



*Ilustración 54 Alta Tarifa: segundo paso*

Una vez el formulario esté rellenado satisfaciendo todas las directrices de obligatoriedad, el sistema permite hacer clic sobre el botón aceptar. Dicho botón al ser presionado genera una petición de alta que envía al servidor backend. El servidor hace los procesos necesarios para dar de alta una nueva empresa con su nueva tarifa asociada en la cual figuran los registros introducidos y comunica el resultado del proceso a la aplicación web la cual muestra el resultado de la operación como se puede observar en la Ilustración 55.

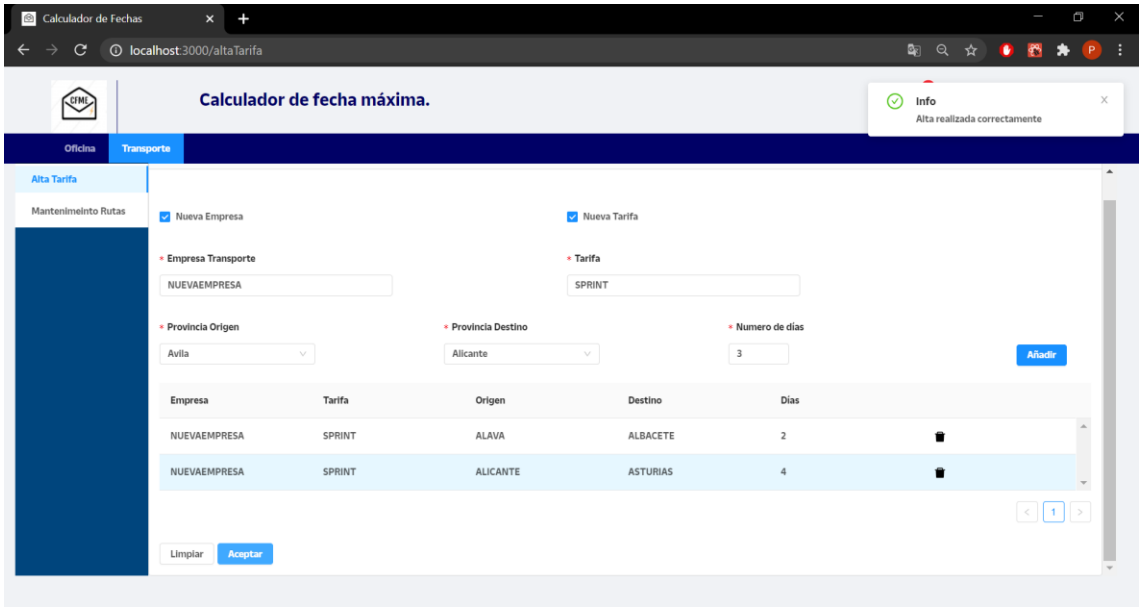



*Ilustración 55 Alta Tarifa: tercer paso*

### 7.2.5 Prueba funcional: Alta Oficina

Para el caso de pruebas relacionado con la página Alta Oficina se va a definir un usuario el cual desea dar de alta una nueva oficina asociada a una empresa cliente existente en la base de datos. Dicha oficina constará de varios horarios de trabajo y un código postal.

El primer paso es abrir el menú horizontal de Oficina y hacer clic sobre el menú vertical en la opción Alta Oficina. El sistema mostrará un filtro con las indicaciones de obligatoriedad para realizar la consulta. Hasta que estas indicaciones no se cumplan el sistema no permitirá hacer clic sobre el botón de Aceptar.

La página de Alta Oficina es la mostrada en la Ilustración 56 que consta de un formulario con los datos necesarios para dar de alta una nueva oficina en la base de datos. Esta nueva oficina puede pertenecer a una empresa ya creada o a una nueva. En caso de ser nueva la empresa a la cual se le crea la oficina también se dará de alta dicha empresa.

El formulario consta de una parte obligatoria y otra opcional. La parte obligatoria es la referente a la localización y tarifa a la que se suscribe la oficina y la opcional al horario de trabajo. El horario es opcional porque el sistema en caso de no tener un horario definido en la oficina aplica un horario estándar de España.

La aplicación web al igual que en el resto de las páginas precarga los combos con la información almacenada en la base de datos, facilitado así la cumplimentación de estos. En este caso la página precarga las empresas cliente y las de transporte para que el usuario seleccione una.

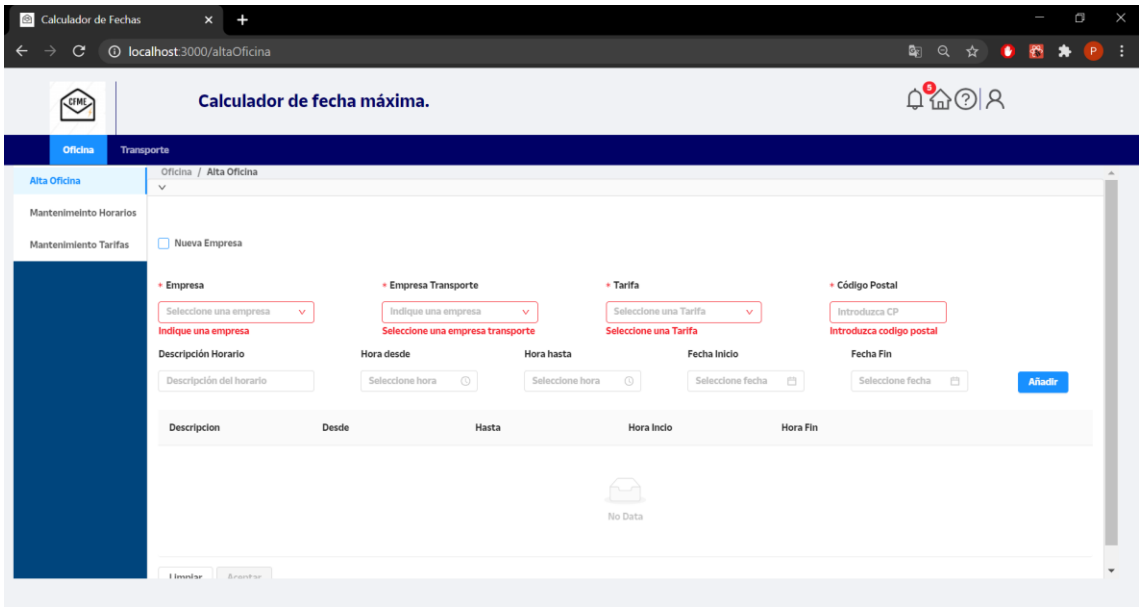



*Ilustración 56 Alta Oficina: estado inicial*

En este caso de prueba el usuario da de alta una oficina la cual pertenece a la empresa cliente Peluquerías Ernesto y esta suscrita a la tarifa Premium de la empresa de transporte Veloz. EL código postal de la oficina será 46900 que hace referencia a la localidad de Torrent en Valencia. La Ilustración 57 muestra el estado de la página con todos los datos introducidos.

Una vez rellenados los campos obligatorios, el sistema permite hacer clic en el botón de Aceptar. Pero como el caso de uso definido incluye el alta de una oficina con varios horarios asignados, el usuario rellenará los combos referentes a estos y hará clic en el botón de Añadir. Este botón eliminará la información de los combos referentes a la información de los horarios y creará una fila en la tabla con los datos introducidos anteriormente como se muestra en la Ilustración 58. Esta fila puede ser eliminada si el usuario no está conforme con los datos introducidos.

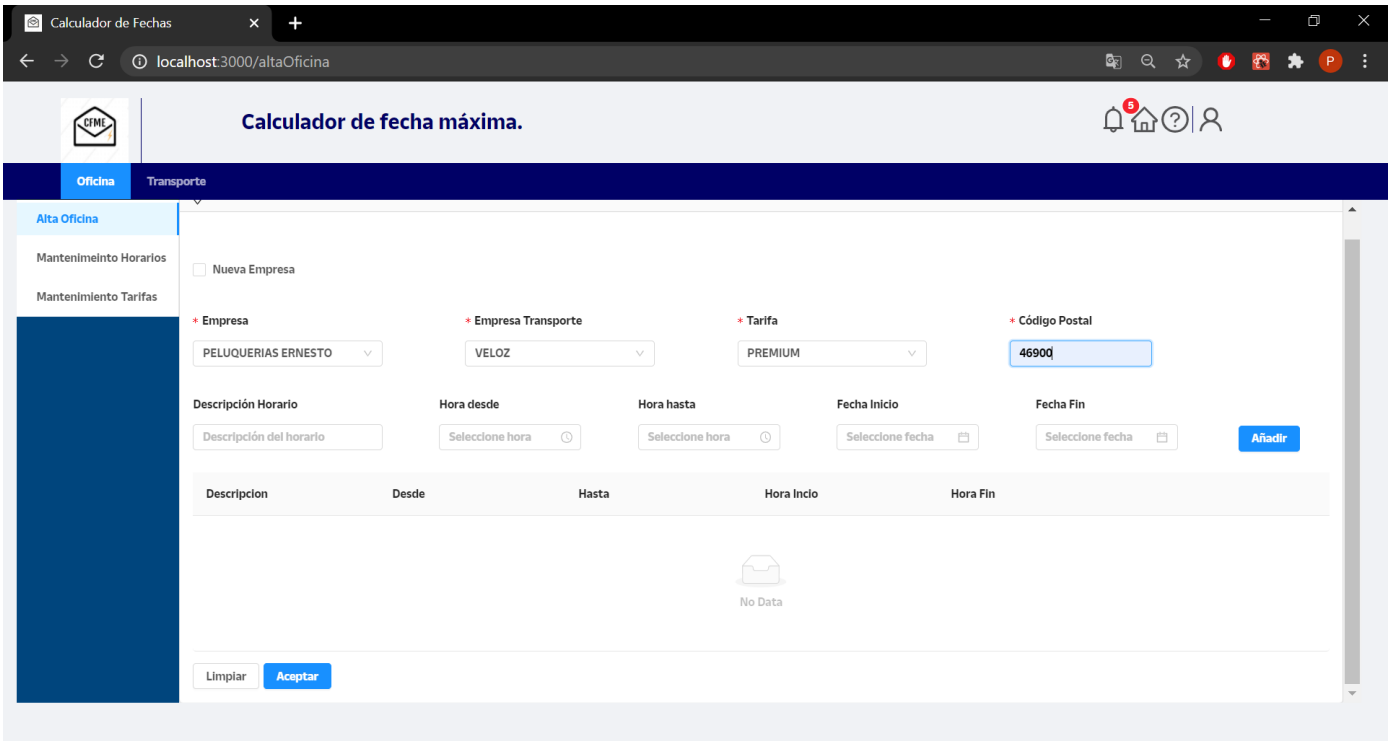



*Ilustración 57 Alta Oficina: primer paso*

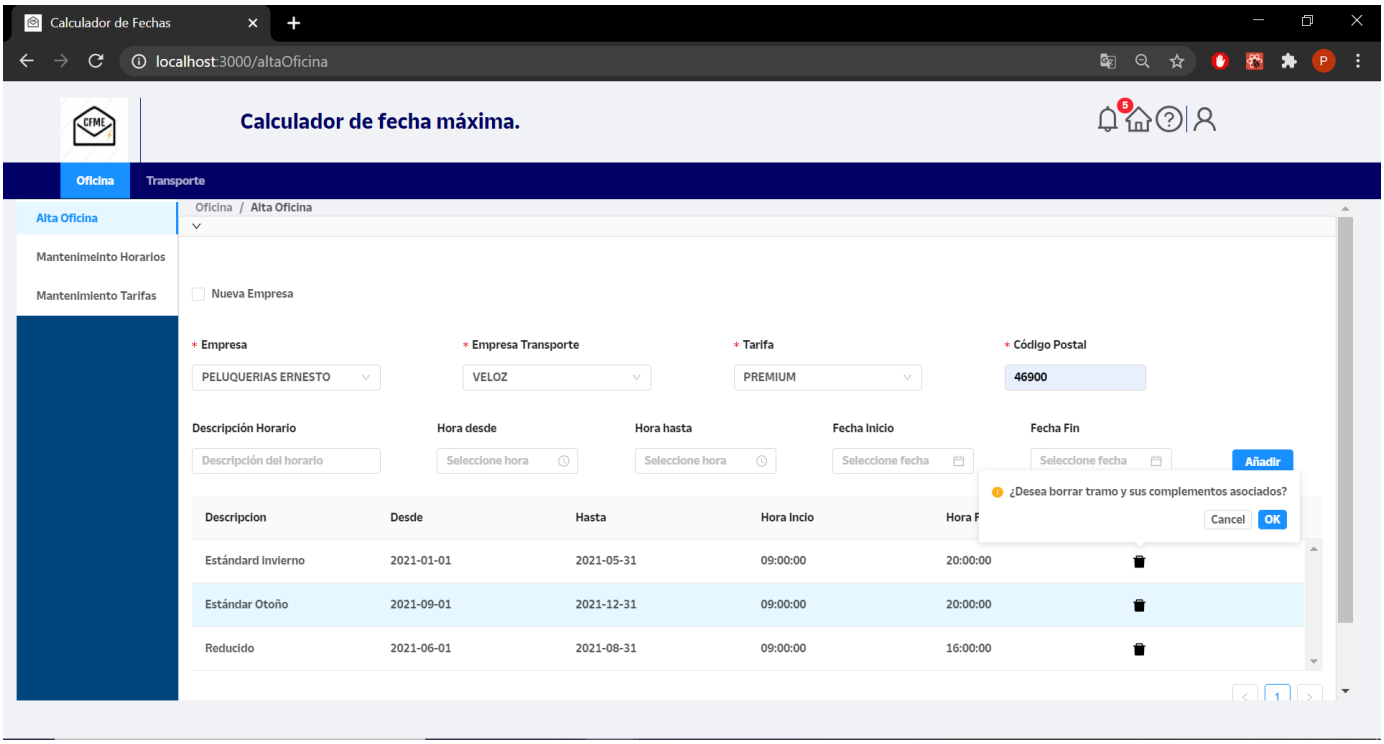



*Ilustración 58 Alta Oficina: segundo paso*

Al igual que la página de Alta Tarifa, una vez el formulario esté rellenado satisfaciendo todas las directrices de obligatoriedad, el sistema permite hacer clic sobre el botón Aceptar. Dicho botón al ser presionado genera una petición de alta que envía al servidor backend. El servidor hace los procesos necesarios para dar de alta una nueva empresa con su nueva tarifa asociada en la cual figuran los registros introducidos y comunica el resultado del proceso a la aplicación web la cual muestra el resultado de la operación como se muestra en la Ilustración 59.

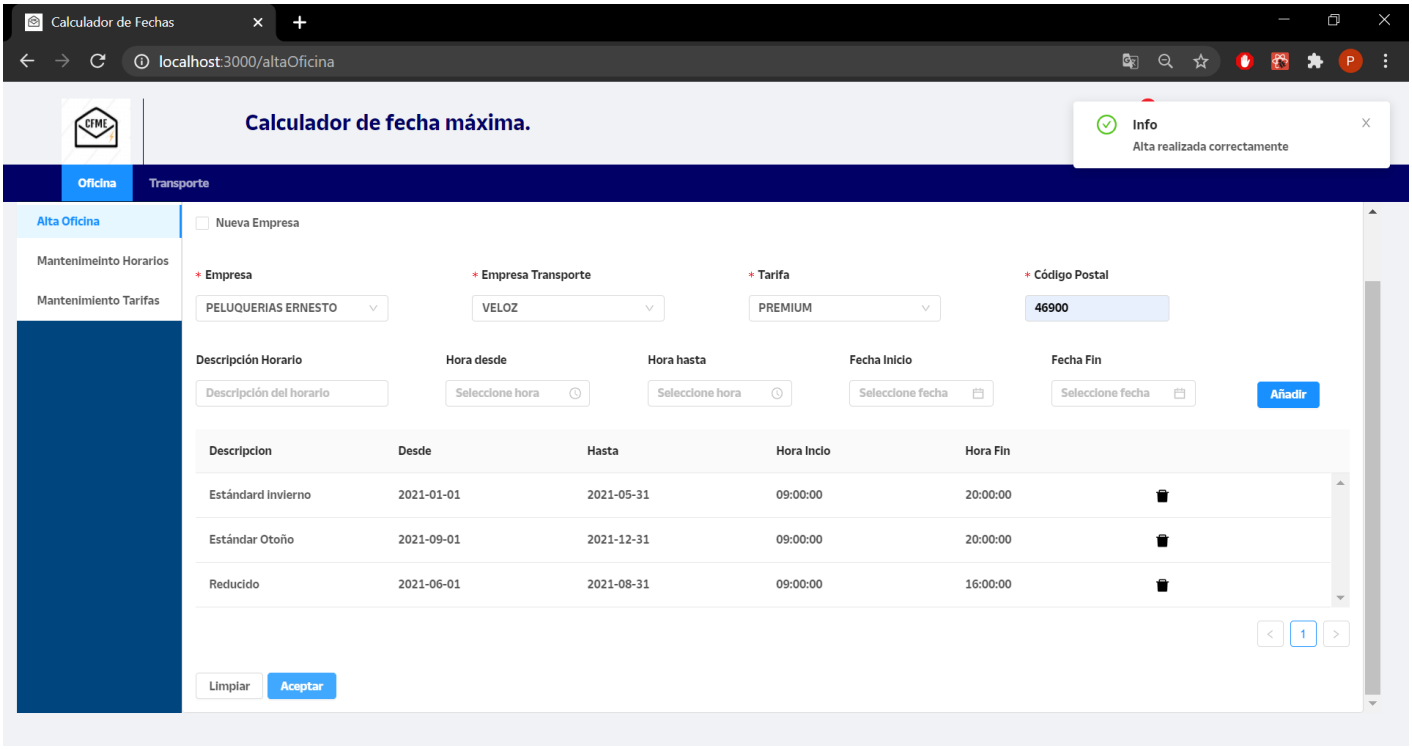



*Ilustración 59 Alta Oficina: tercer paso*

# 8. Conclusiones

Se ha visto como el mercado cada vez demanda más las arquitecturas orientas a servicios y se desmarca de las arquitecturas monolíticas. Por ello se ha elegido diseñar un servicio dentro de este contexto para aportar una nueva funcionalidad muy valiosa en el ámbito de la oferta de productos vía online.

El proyecto desarrollado ha permitido obtener un sistema que genera una fecha de entrega máxima personalizable y accesible desde todas las plataformas. Se ha desarrollado con las tecnologías definidas en el diseño y el resultado cumple con los objetivos y los requisitos especificados.

Bien es cierto que el sistema funciona con datos funcionales insertados a mano y no son los reales. El volumen de la base de datos no es el que en realidad debería ser, pero las consultas y procesos de generación de respuestas son ágiles y fiables.

Estas características se pueden ver afectadas en un contexto de producción con demoras en el acceso de bases de datos, pero se puede solucionar fácilmente con la inserción de una memoria cache que agilice aún más los accesos.

Por otro lado, la aplicación web diseñada contiene todas las funcionalidades necesarias para que los usuarios puedan emplear el servicio web adaptado a sus necesidades.

En conclusión, en este trabajo se ha creado un sistema de generación de fechas actualizado, con tecnologías en uso, con una cierta fiabilidad, personalizable y accesible para cualquier empresa o usuario.

## 8.1 Relación del trabajo desarrollado con los estudios cursados

Durante mi estancia en la Universidad he adquirido numerosas destrezas que me han permitido el desarrollo de este proyecto.

Dichas destrezas abarcan campos, contenidos en este proyecto, como; la generación, configuración y manipulación de bases de datos aprendido en la asignatura de Bases de Datos y Sistemas de la Información (BDA) y Tecnología de Bases de Datos (TBD); la implementación y configuración de servicios web aprendido en Integración de Aplicaciones (IAP); el levantamiento, despliegue y puesta en marcha de un servidor en la red, aprendido en la asignatura Sistemas y Servicios en la Red (SSR); y la

creación, organización e implementación de una aplicación aprendido en Ingeniero del Software (ISW), Introducción a la Programación (IAP) y Programación (PRG).

Todos estos campos nombrados se pueden encontrar en el proyecto en la creación y configuración de la base de datos, la creación de la aplicación que generará la fecha de entrega máxima de paquetes, la organización REST implementada y la conexión entre los dos servidores que conforman el sistema.

# 9. Trabajos futuros

Para finalizar se exponen a continuación algunos trabajos que podrían mejorar la aplicación desarrollada.

Lo primero, cabe comentar que los datos con los que se ha trabajado son datos funcionales para probar el correcto funcionamiento del sistema, no son el volumen real con el que trabajaría un sistema en producción. Para que el sistema esté en un alto rendimiento en un contexto de producción, habría que rehacer las pruebas de características del servicio y en caso de dar unos tiempos mayores, sería necesario la instalación de una memoria cache.

En cuanto a seguridad se refiere el proyecto no ha desarrollado ningún mecanismo, pero el propio *Spring-framework* que se ha usado permite la configuración de roles para la identificación antes del uso del servicio. Por lo que para una correcta seguridad sería necesario: configurar dicho *frameworck* con roles que permitan solo alterar los datos propios del usuario y consultar nada más que los de otros; crear una página que funcione a modo de identificación para que cada usuario reciba en ella un token con su rol; y configurar un servidor de autenticación como podría ser un LDAP con los usuarios dados de alta en el servicio.

Aunque el sistema es plenamente funcional con altas y eliminaciones, lo óptimo sería añadir la edición para agilizar la interacción del usuario con los procesos de modificación de datos. Es decir, para modificar un dato ahora en la aplicación es necesario eliminar el registro actual y dar de alta uno nuevo con los datos modificados.

Por último, el sistema solo trabaja a nivel nacional con lo cual está bastante restringido al uso. Una futura mejora podría ser ampliar la lógica para este trabajase a nivel internacional y así conseguir un calculador de fecha de entrega máxima globalizado y centralizado.

# 10. Referencias

[1] Bowman, J. P. (1996). The Digital Economy: Promise and Peril in the Age of Networked Intelligence. *Academy of Management Perspectives*. https://doi.org/10.5465/ame.1996.19198671

[2] Fallis, A. . (2013). Economía digital para el cambio estructural y la igualdad. *Journal of Chemical Information and Modeling*. https://doi.org/10.1017/CBO9781107415324.004

[3] Mesenbourg, T. L. (2001). Measuring Electronic Business. *U.S. Bureau of the Census*.
(Bowman, 1996; Fallis, 2013; Mesenbourg, 2001)

[4] Kamlesh, K. (2020, junio 19). The Spring AplicationContext. Retrieved from https://www.baeldung.com/spring-application-context

[5] Rodríguez, A. (2015, febrero 09) Servicios de RESTful: Los aspectos básicos. Retrieved from https://developer.ibm.com/es/articles/ws-restful/#:~:text=REST%20define%20un%20conjunto%20de,est%C3%A1n%20escritos%20en%20diferentes%20lenguajes.

[6] Liu, P. (2015, mayo 22) Building RESTful Web Service in NetBeans. Retrieved from https://netbeans.org/community/magazine/html/04/restfulws.html

[7] Nielsen, J. (2004). Usability engineering. In *Computer Science Handbook, Second Edition*. https://doi.org/10.1201/b16768-38

# 11. Glosario

1. API: acrónimo de *Application Programing Interface*; es un enlace entre diferentes componentes software.
2. Backend: división de las tareas de programación, en específico corresponde a las tareas relacionadas con la gestión de las comunicaciones entre distintos componentes.
3. Frontend: división de las tareas de programación, hace referencia a las actividades relacionadas con las estructuras visuales del contenido web.
4. Servlet: clase de un programa JAVA que se utiliza para la ampliación de las funcionalidades de un servidor.
5. Debugging: término que se refiere a la depuración del código en búsqueda de errores para su posterior solución.
6. Framework: es una estructura conceptual y tecnológica que sirve de soporte para proyectos software.
7. Bean: son objetos JAVA que el sistema de Spring maneja para que la propia clase no tenga que crearlos.
8. EAR: extensión con la que se guardan los archivos ejecutables de programas desarrollados con JAVA EE.
9. XML: acrónimo de *Extensible Markup Lenguage*, formato para la comunicación entre dispositivos que permite diferenciar entre objetos, sus propiedades y valor a través de marcas.
10. JSON: acrónimo de *JavaScript Object Notation*, es un formato estandarizado que permite distinguir su contenido mediante el empleo de llaves y corchetes.
11. Plug-in: hace referencia a una aplicación que se complementa con otra para ofrecerle nueva funcionalidad y generalmente muy específica.
12. URI: acrónimo de *Uniform Resource Locator* es un localizador de recursos uniforme que identifica recursos de forma variante en el tiempo.
13. REST: acrónimo de Representational State Transfer es un tipo de organización basada en URIs.
14. Arquitectura a tres niveles: especialización de la arquitectura cliente-servidor en la que se definen tres capas. La de presentación donde se ubica la interfaz, la de negocio donde se realizan los cálculos y la de persistencia donde se realizan los accesos a la base de datos.

# 12. Apéndices

## 12.1 Modelo de Datos