UNIVERSIDAD POLITECNICA DE VALENCIA

Escuela Técnica Superior de Ingeniería del Diseño

Diseño e implementación de una aplicación web y adaptación de una API para un portal de gestión de contraseñas de varias plataformas empresariales

TRABAJO FINAL DE GRADO

Grado en Ingeniería Electrónica Industrial y Automática

Autor: Óscar Sánchez Cutillas

Tutor: Carlos Domínguez Montagud

Valencia, septiembre de 2019

Índice de contenidos

Índice de figuras

Índice de tablas

Glosario

API – Interfaz de programación de aplicaciones (del inglés *Application Programming Interface,* API), conjunto de funciones que exponen públicamente una serie de servicios implementados en una librería, para ser utilizados por terceras partes.

Backend – Parte de la infraestructura de una empresa encargada de procesar y almacenar la información.

Web – En este contexto, que está alojado en la *World Wide Web* (WWW) o relacionado con la misma.

Software – Conjunto de programas, instrucciones y reglas informáticas para ejecutar ciertas tareas en una computadora.

Scripting, script – Conocido como guion o archivo de procesamiento por lotes, son archivos que contienen un conjunto de instrucciones ejecutables, suelen ser interpretadas.

Outsourcing – Conocido en español como subcontratación, proceso económico empresarial en el que una sociedad mercantil transfiere los recursos y las responsabilidades referentes al cumplimiento de ciertas tareas a una sociedad externa.

Frontend – Parte de la infraestructura de la empresa encargada de la presentación del contenido y la información al usuario en las plataformas que éste requiera.

Framework – Constituye un conjunto de librerías, patrones de diseño y prácticas que permiten la resolución de un problema concreto.

Evergreen – En el contexto de los navegadores *web*, engloba a todos aquellos que actualizan sus motores de navegación automáticamente, sin la intervención del usuario. Así, se asegura que todos aquellos usuarios que utilicen esos navegadores podrán visualizar correctamente las páginas *web* diseñadas con las tecnologías más recientes soportadas por los mismos.

URL – Del inglés *Universal Resource Locator*, es una secuencia de caracteres estandarizada que permite a un navegador *web* encontrar un recurso concreto en un servidor.

I
Memoria

1 Introducción

1.1 Objeto del trabajo

El objeto de este trabajo es el diseño e implementación de un portal de gestión de contraseñas para varias plataformas empresariales. Esta aplicación sustituirá a otra ya existente, desarrollada en una tecnología ya obsoleta. El trabajo consta de dos partes:

- Diseño e implementación de una aplicación *web* que permita realizar todos los cambios necesarios en las contraseñas de las cuentas de los usuarios de las distintas plataformas empresariales.
- Diseño e implementación de las adaptaciones necesarias a las *APIs* y bibliotecas existentes en el *backend* de la empresa, para dar servicio a la aplicación *web*. Estas adaptaciones permitirán llevar a cabo el proceso de introducción, cambio y recuperación de las contraseñas de las cuentas de los usuarios de manera segura, así como de proveer toda la información que necesite la aplicación *web* (reglas para la introducción de contraseñas, traducciones y colores e imágenes corporativas).

1.2 Motivación

Una de las principales motivaciones que me llevó a realizar este trabajo fue la de crear una aplicación web empresarial bajo la supervisión de la empresa para la que hice prácticas (ODEC Centro de Cálculo y Aplicaciones Informáticas, SA). Gracias a esto, he podido aprender a desarrollar una aplicación web en un entorno real, enfrentándome a problemas reales y aplicando patrones de diseño, buenas prácticas y utilizando herramientas habituales de empresas del sector.

Todo este conocimiento adquirido me ha permitido mejorar mi formación en el sector de la informática y el desarrollo de *software*, lo cual supone una ventaja de cara a mi futuro laboral.

1.3 Estructura del documento

Este documento está estructurado en capítulos que siguen el orden con el que se ha concebido y desarrollado este proyecto. Dado que existen dos grandes bloques en este trabajo, la aplicación *web* y las adaptaciones al servidor, se ha enfatizado en la estructura del documento esta dicotomía, siguiendo un esquema predecible, donde se tratan primero los temas de ese capítulo relacionados con el bloque de la aplicación *web* y, a continuación, aquellos relacionados con el bloque de las adaptaciones al servidor. En algunos casos, se ha desgranado cada bloque en un capítulo separado.

En el segundo capítulo se describe brevemente el contexto tecnológico del sector y se introduce a la empresa para la que se lleva a cabo este proyecto, mostrando la implementación actual de la funcionalidad que se va a desarrollar.

En el tercer capítulo se explican las tecnologías que componen la aplicación web y el servidor y las características que las hacen apropiadas.

En el cuarto capítulo se detallan todas las herramientas utilizadas para el desarrollo del *software* y la redacción de este documento, incidiendo en los motivos de su elección.

En el quinto capítulo puede encontrarse los requerimientos estéticos y funcionales que deben cumplirse tanto en la aplicación *web* como en las adaptaciones a las *APIs* y librerías del servidor.

En los capítulos sexto y octavo se pormenorizan los procesos de diseño y desarrollo de la arquitectura y los distintos componentes que forman la aplicación *web*, así como las decisiones que se han tomado y patrones de diseño que se han implementado.

En los capítulos séptimo y noveno se detallan los procesos de diseño y desarrollo de las adaptaciones necesarias a las distintas capas de las *APIs* existentes en la infraestructura de la empresa.

Por último, en el capítulo décimo se exponen las conclusiones que han podido extraerse de la ejecución de este trabajo y las posibles mejoras al sistema de cara al futuro.

2 Contexto

2.1 Últimos avances en el sector

Una buena forma de comprender el funcionamiento y las necesidades, tanto de la aplicación *web* como de las librerías y *APIs* del servidor, es conociendo su origen, su evolución y los avances que les permitieron cada vez ir ofreciendo una mejor experiencia al usuario.

Hasta mediados de la década de 1990, los desarrolladores que deseaban crear una aplicación debían programarla para ser ejecutada en cada uno de los distintos sistemas operativos en los que sus usuarios la instalaran [1]. Las aplicaciones eran altamente dependientes de los sistemas operativos y de la arquitecura de éstos, dilatando enormemente los tiempos de desarrollo y pruebas (*testing*) de las aplicaciones.

A finales de 1994 se publica la primera versión del navegador Netscape, conocido como el primer navegador *web* comercial de la historia. A partir de este momento, el público general comenzó a tener acceso a Internet y a la *World Wide Web* (WWW) y comenzó su adopción masiva.

El usuario accedía a un dominio *web*, descargaba un documento HTML y visualizaba de manera estática el contenido que había descargado.

En 1995, Netscape presentó JavaScript, un lenguaje de *scripting*, que permitió la creación de páginas *web* dinámicas, donde ya no era necesario volver a cargar la página *web* cada vez que se producía un cambio en la interfaz.

La aparición de Macromedia Flash en 1996 supuso un revolución, posibilitando el enriquecimiento de las páginas *web* con animaciones, gráficos vectoriales y videojuegos embebidos.

Poco después, en 1997 [2], se estandarizó el protocolo CGI (*Common Gateway Interface*), que posibilitaba el envío de páginas *web* personalizadas. De ese modo, no todos los usuarios tenían que recibir necesariamente el mismo contenido, sino que éste podía ser personalizado [3]. Este protocolo ofrece una interfaz común con la que los servidores *web* podían ejecutar un programa externo con los argumentos recibidos por parte del cliente. Este método, aunque conveniente fue cayendo en desuso por sus problemas de rendimiento.

Sin embargo, no fue hasta 2005 con la aparición del conjunto de técnicas de desarrollo conocidas como AJAX (*Asynchronous JavaScript And XML*), y que más tarde se traducirían en la publicación de la librería jQuery [4], que se facilitó la creación de potentes experiencias interactivas y se dotó a las páginas *web* la posibilidad de comunicarse de manera asíncrona con su servidor. A partir de aquí podemos comenzar a hablar de aplicaciones *web*, donde dejan de ser necesarios los refrescos de pantalla y la aplicación es capaz de realizar peticiones al propio

servidor sin interferir en la experiencia de uso. Es entonces, cuando las antiguas páginas *web*, ahora aplicaciones *web*, pudieron comenzar a rivalizar con las aplicaciones de escritorio.

Dado que todos los usuarios disponían de navegador *web* en sus respectivos sistemas operativos, para muchas aplicaciones ya no era necesario desarrollar varias implementaciones para cada uno de estos sistemas operativos, sino desarrollar una sola aplicación *web*, reduciendo notablemente el tiempo de desarrollo y pruebas de las aplicaciones.

Desde entonces, se han desarrollado diversas tecnologías como Angular y React, con tasas de uso en 2019 de 30.7% y 31.3% respectivamente [5], que permiten la creación de aplicaciones web muy complejas, con interfaces refinadas e infraestructuras escalables, creando nuevos modelos de negocio y nuevos productos.

2.2 Acerca de ODEC

ODEC Centro de Cálculo y Aplicaciones Informáticas, SA es la empresa para la que se ha realizado el desarrollo del *software* que compone este trabajo de final de grado.

ODEC es una empresa con más de 50 años de experiencia [6] con sedes en Gandia, Valencia, Madrid y Barcelona y presta servicios informáticos especializados en la captura de datos, tratamiento de información, presentación de resultados, desarrollo de *software* y *outsourcing* de servicios.

Figura 1: Logotipo de la empresa

Entre los principales sectores para los que ODEC trabaja destacan: investigación de mercados, medios publicitarios, administraciones públicas, marketing, automoción y elecciones.

La funcionalidad que se va a implementar en este trabajo forma parte de Cemtric, un producto creado por ODEC para el sector de la automoción. Cemtric es una plataforma de gestión de satisfacción del cliente utilizada por diversas compañías automovilísticas a nivel mundial, como Audi, Renault y Volkswagen, entre otras.

2.3 Implementación actual de las funcionalidades

Actualmente, cada uno de los clientes que contratan el servicio Cemtric disponen de una plataforma personalizada según sus necesidades. Sin embargo, dado que todas ellas realizan el mismo tipo de operaciones sobre las contraseñas de las cuentas de sus usuarios, se consideró oportuno hacer una única aplicación web centralizada, reduciendo el tiempo de desarrollo y mantenimiento con respecto al desarrollo una aplicación para cada cliente.

La aplicación se genera íntegramente en el servidor que la provee, volviendo a generarse en cada cambio de pantalla, y está implementada a nivel de frontend en Java con la tecnología JSF (Java Server Faces) y a nivel de backend está implementada en Java.

En esta aplicación, anteriormente mencionada el usuario puede llevar a cabo los siguientes procesos:

- Introducción de la primera contraseña tras la creación de la cuenta del usuario.

- Cambio de la contraseña antigua, conocida por el usuario, por una contraseña nueva.
- Petición de correo electrónico de recuperación de contraseña (contraseña olvidada).
- Introducción de una contraseña nueva, en los casos en los que el administrador fuerce al usuario a cambiar su contraseña, por motivos de seguridad.
- Introducción de una contraseña nueva, sin conocer la anterior, tras ser redirigido desde un correo electrónico con un enlace de recuperación de contraseña.

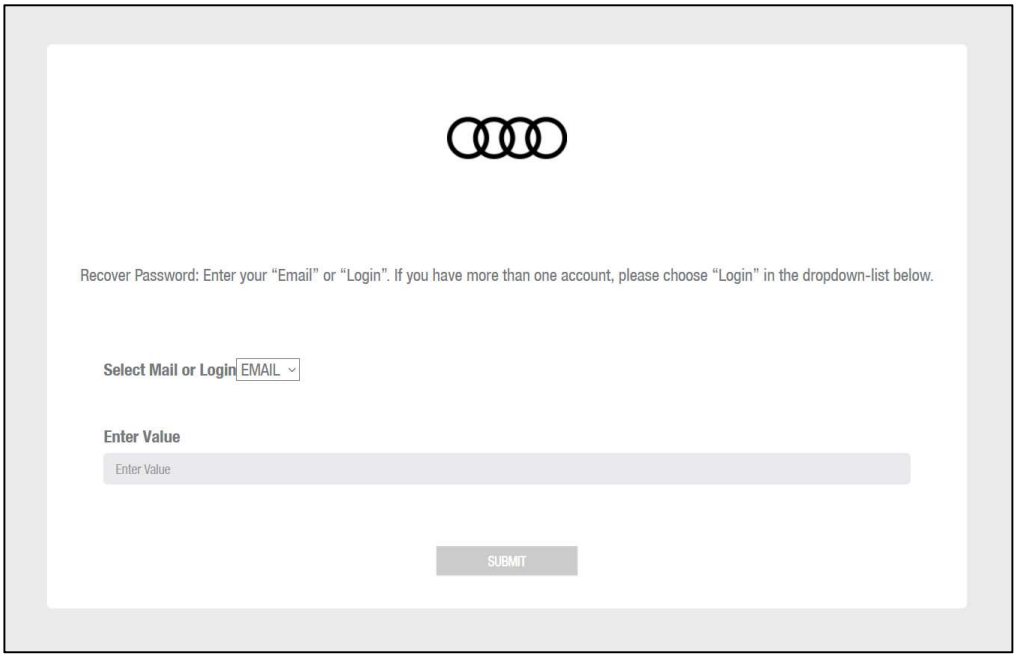

Figura 2: Implementación actual de la aplicación web

La nueva implementación de la aplicación web mantendrá estas funcionalidades y las reprogramará en las nuevas tecnologías utilizadas, solventando defectos de los que adolecía la anterior implementación como:

- Falta de adaptación de la interfaz de usuario a nuevos dispositivos móviles como las tabletas o los teléfonos móviles inteligentes.
- Falta de imágenes y colores corporativos.
- Falta de traducciones a otros idiomas distintos al inglés.
- Actualización de la interfaz y mejoras a nivel estético.
- Uso de tecnologías obsoletas.

Con respecto a las adaptaciones a las APIs y librerías en el *backend*, las principales mejoras a realizar son:

- Publicación de los métodos existentes que la aplicación *web* necesite a través de la arquitectura REST.
- Creación y publicación de nuevos métodos necesarios.
- Adaptaciones a otras librerías para poder llevar a cabo los puntos anteriores.

3 Tecnologías empleadas

Las tecnologías empleadas tanto en el desarrollo de la aplicación *web* como de los programas alojados en los servidores dependen exclusivamente de la política de la empresa, ODEC. Por tanto, en este apartado no se discutirán las razones del uso de estas tecnologías, sino que se indicarán al lector las razones por las que la industria las considera apropiadas para su uso en el desarrollo de la aplicaciones *web* y en el desarrollo de servidores.

3.1 Aplicación web

- TypeScript: TypeScript es un superconjunto, o extensión de JavaScript [7]. JavaScript es uno de los lenguajes más importantes de Internet, soportado por todos los navegadores y utilizado en mayor o menor medida por la mayoría de las páginas y aplicaciones *web* alojadas en Internet. A nivel técnico, JavaScript destaca por ser un lenguaje interpretado, con tipado dinámico de las variables y multiparadigma. Así pues, TypeScript reúne todas las características de JavaScript y añade otras características que lo hacen más seguro para el desarrollo empresarial de aplicaciones *web* como:
  - La posibilidad de declarar en el código los tipos de las variables.
  - Nuevos tipos, que permiten manipular la información con mayor seguridad.
  - Interfaces para declarar la estructura de los objetos.
  - Enumeraciones, que permiten restringir el conjunto de valores posibles que puede almacenar una variable.

  Cabe destacar que TypeScript no es un lenguaje interpretado como JavaScript, es un lenguaje compilado. TypeScript se compila, antes de ser usado, a JavaScript y, por tanto, es compatible con todas las librerías desarrolladas en JavaScript y puede ser utilizado en cualquier navegador o entorno que soporte JavaScript.
- HTML: HTML (*Hypertext Markup Language*) es un lenguaje de marcado que permite definir la estructura de un documento, en este caso, una aplicación *web*. Se considera un estándar en la industria.
- SASS: Es un lenguaje de diseño gráfico que extiende a CSS (*Cascade Style Sheets*), que posibilita la adición de estilos personalizados al contenido estructurado en HTML. La principal diferencia de SASS con CSS es que proporciona una mayor facilidad para estructurar y jerarquizar el código, creación de variables y uso de *mixins* para reutilizar el código en otros lugares.

- NodeJS: Es un entorno de ejecución de código JavaScript fuera de un navegador creado con el motor de ejecución V8 de Chrome. Es la base sobre la que funcionan librerías y *frameworks* como React o Angular, respectivamente. Viene instalado con npm (node package manager), uno de los mayores gestores de paquetes JavaScript, ampliando enormemente las posibilidades de ampliación y mejora de Angular.
- Angular: Angular es un *framework* de código libre que destaca por proporcionar un entorno con el que desarrollar aplicaciones *web* de página única (*Single-Page App*). Angular está diseñado para ser utilizado con TypeScript, HTML y SASS (también permite CSS o alguna extensión de este). Dispone de múltiples librerías para resolver diversos problemas relacionados con el diseño de aplicaciones *web* de página única como las rutas, conexiones a servidores, carga bajo demanda, pruebas (*testing*), entre otros. Está desarrollado por Google, lo cual garantiza periodos fijos de mejoras y soporte.

3.2 Servidor

En el caso de las tecnologías utilizadas en el servidor, dado que su uso se comparte con otros productos empresariales, el conjunto de tecnologías usadas en éstos excede ampliamente el area de conocimiento que se abarca en este documento. A continuación se explicarán aquellas tecnologías con las que se haya tenido que interactuar directamente:

- Java (JEE): Es un lenguaje de programación, compilado, de propósito general y orientado a objetos. Es ejecutado en una máquina virtual, por lo que los programas creados en Java pueden ejecutarse en cualquier sistema operativo que disponga de una máquina virtual Java. Su sintaxis permite la creación de programas empresariales muy robustos, estructurados y duraderos. Dentro Java, se hace uso de un framework proporcionado por la especificación JEE. JEE proporciona los mecanismos de acceso a recursos y transacciones y comunicaciones.
- SQL: Es un lenguaje de consulta estructurada ampliamente extendido que permite la introducción, modificación y eliminación de contenido en bases de datos relacionales. En el caso de este proyecto, se utilizará principalmente para la obtención de datos con los que realizar pruebas a las adaptaciones al servidor.

4 Herramientas empleadas

En este apartado, se detallan las herramientas utilizadas para el desarrollo del código de este proyecto y para la redacción de este documento.

4.1 Herramientas para el desarrollo de la aplicación web

- Visual Studio Code: Es uno de los editores de código más populares para el desarrollo de aplicaciones web. Entre sus principales características se encuentran:
  - Es de código libre y gratuito.
  - Ampliable con un gran número de extensiones para facilitar la edición de código.
  - Autocompletado de código, consola integrada, reconocimiento y resaltado con colores de los distintos lenguajes de programación.
  - Interfaz sencilla y consume pocos recursos del ordenador.
  - Posee una consola de comandos (CMD) integrada.
- Firefox, Chrome, Edge, Safari, Internet Explorer 11: Estos navegadores son los más utilizados por los usuarios que se conectan a Internet [8]. Dado que la aplicación web va a ser consumida desde todos estos navegadores, ha sido necesario comprobar que en ninguno de ellos se producen errores, aunque el navegador Firefox ha sido el principal navegador utilizado durante el desarrollo. Entre los motivos del uso de Firefox como navegador principal destacan su naturaleza de código abierto, su velocidad y su uso como navegador habitual.
- Git, GitLab: Git es el programa de control de versiones de código más popular entre los desarrolladores. Usado como estándar en la industria, destaca por ser ligero, eficaz y sencillo de utilizar. Sin embargo, Git se utiliza a través de una consola de comandos, por lo que, por comodidad, se gestiona a través de una interfaz de usuario integrada en Visual Studio Code. Cabe destacar que Git es un programa que almacena las versiones del código en el propio ordenador, y que, si se desea compartir ese código con otros desarrolladores, se debe recurrir a servicios externos (que pueden ser gratuitos o de pago) como GitLab, que es el que utiliza la empresa.
- Angular CLI: Se trata de una herramienta de código abierto que se utiliza a través de una interfaz de linea de comandos y automatiza la generación de proyectos y de código para aplicaciones desarrolladas en Angular.

4.2 Herramientas para el desarrollo de las adaptaciones al servidor

- Eclipse IDE: Con soporte para Java, entre otros, es un potente entorno de desarrollo integrado (IDE) que permite la creación, programación y mantenimiento de proyectos empresariales en Java, SQL, XML y muchos otros lenguajes, y dispone de herramientas como:
  - Depurador de código.
  - Gestior de dependencias.
  - Autocompletado avanzado de código.
  - Gestor de versiones con Git, Subversion, etc.
  - Instalador de complementos o extensiones.
  - Compilación y ejecución de código Java en máquinas virtuales.
- Swagger: Es una herramienta que, a partir de una serie de anotaciones en el código, genera una página *web* en la que el desarrollador puede visualizar e interactuar con los recursos de una *API* diseñada a partir de la arquitectura *REST*. Dicha página *web* puede ser visualizada desde cualquier navegador que esté conectado a la red en la que se genera la página.
- SQL Developer: Es un entorno de desarrollo integrado (IDE) diseñado para la edición y ejecución de código SQL, utilizado para la gestión de bases de datos relacionales.

4.3 Herramientas utilizadas para la redacción de este documento

- Microsoft Word: Es el programa principal con el que se ha redactado este documento. Utilizado ampliamente en muchos ámbitos, posibilita la creación de documentos con multitud de opciones de formato y autogeneración de contenidos, como tablas de contenido, índices y bibliografías, entre otras.
- Draw.io: Es una herramienta online para la creación de bocetos de la interfaz gráfica de usuario. Dispone de controles sencillos de utilizar y genera bocetos de gran calidad en diversos formatos.
- Lucidchart: Es una herramienta online de creación de diagramas. Permite la creación de diagramas siguiendo el lenguaje de modelado UML.

5 Requisitos

A continuación se muestran los requisitos para esta nueva implementación, tanto a nivel estético como a nivel funcional.

5.1 Aplicación web

- Implementada con el framework Angular en su versión más reciente.
- Diseñada para ser visualizada en todo tipo de dispositivos como teléfonos inteligentes, tabletas, portátiles y ordenadores de escritorio.
- Debe soportar navegadores *evergreen*, como Firefox, Chrome, Safari, Edge, entre otros. Destaca también la necesidad de soportar Internet Explorer a partir de la versión 11.
- Debe detectar el idioma preferido del usuario y mostrarle el contenido en él. En caso de ser el idioma equivocado, debe permitir al usuario cambiar el idioma.
- La interfaz debe ser limpia y moderna.
- La aplicación debe tener el menor tamaño posible.
- La aplicación debe funcionar de forma fluida en la mayoría de dispositivos.
- La aplicación debe permitir al usuario llevar a cabo los procesos descritos en el apartado 2.3.
- La interfaz debe incluir el logo de la compañía, colores corporativos y, opcionalmente, una imagen de fondo.
- Desarrollada con buenas prácticas y de la forma más óptima y estructurada.

5.2 Adaptaciones al servidor

- El código debe ser debidamente documentado.
- Todas los posibles problemas que impidan a una función devolver aquello que se espera de ella deben emitir una excepción, para ser contemplados en otras partes del código.
- Todas las peticiones al servidor deben recibir respuestas con códigos HTTP, que expresen, de la forma más correcta, la naturaleza del contenido devuelto en la respuesta.

- Salvo casos estrictamente necesarios, todas las comunicaciones con la aplicación *web* deben ser seguras, evitando el acceso de personas no autorizadas a los medios del servidor.
- Debe proveer las traducciones al usuario que éste precise y, en caso de no haberlas, debe entregar los textos en inglés.
- Debe disponer públicamente de todos los métodos que requiera la aplicación *web* para llevar a cabo todos los procesos de introducción y cambio de contraseñas así como el envío de correos electrónicos de recuperación.
- Todas las peticiones deben resolverse de manera consistente en periodos de tiempo breves y que supongan la menor molestia posible al usuario.

6
Diseño de la aplicación web

Antes de detallar la arquitectura de la aplicación, es conveniente hacer una breve introducción explicando las partes fundamentales que componen una aplicación en Angular.

Angular se basa en la creación de Componentes, que son pequeñas porciones de código cuya función es mostrar contenido en pantalla. Dichos componentes pueden contener elementos estáticos, elementos dinámicos o ambos. Cuando se crea un componente con la herramienta Angular CLI (explicado más adelante), se crean cuatro archivos:

- nombre.component.html: Contiene la estructura de los elementos visuales del componente.
- nombre.component.scss: Contiene los estilos de los elementos visuales del componente.
- nombre.component.ts: Contiene toda la lógica del componente. Aquí es donde se obtiene la información visualizada en los elementos visuales del componente y es donde se reacciona a eventos producidos en la interfaz.
- nombre.component.spec.ts: Contiene las pruebas (*tests*) que se utilizan para comprobar el correcto funcionamiento del componente.

Para proveer de datos a estos componentes, Angular permite la creación de Servicios, encargados de realizar las peticiones necesarias a las *APIs* o de extraer aquella información guardada en otras partes del código. Los servicios también permiten sincronizar información entre dos o más componentes.

Los componentes y/o servicios creados se agrupan en Módulos, que permiten separar conceptualmente las distintas partes de la aplicación. La organización del código en módulos permite, en los casos necesarios, su cargar bajo demanda (*lazy*), evitando la carga de módulos que no se vayan a utilizar durante los primeros momentos de uso de la aplicación y reduciendo el tiempo de carga de la aplicación.

Los módulos pueden incluir opcionalmente un fichero del tipo *nombre-routing.module.ts* que contiene una relación de direcciones *web* o rutas y componentes. De este modo, puede establecerse que el componente «Bienvenida» sea visible cuando el usuario o la aplicación entren en la dirección *https://www.compañia.com/bienvenida*. De este modo, se pueden incluir y anidar módulos dentro de otros módulos (ej. *https://www.compañia.com/bienvenida/primeros-pasos*).

Las rutas que se consideren oportunas pueden protegerse con el uso de *guards*, que son clases adheridas a la interfaz *CanActivate* de Angular y que interceptan los componentes antes de ser visualizados, permitiendo al desarrollador realizar comprobaciones, normalmente de seguridad, e impidiendo al usuario el acceso a este componente si las comprobaciones no se cumplen.

6.1 Arquitectura de la aplicación

El código que compone esta aplicación está organizado en cuatro grandes módulos: *Core*, *Static*, *Features* y *Shared*.

El propósito de segmentar el código en estos cuatro módulos es la de poder separarlo conceptualmente y a nivel de responsabilidades. El módulo Shared es completamente exportable a cualquier otro proyecto de la empresa, reduciendo los tiempos de desarrollo y pruebas. El módulo Core puede contener código reutilizable, aunque no todo. En el caso de los módulos Static y Features, el código es demasiado específico y difícilmente exportable.

Es importante destacar que, tanto a nivel de *frontend* como a nivel de *backend*, todo el proceso de introducción, cambio y recuperación de contraseña va ligado a la existencia de un *ticket*, almacenado en la base de datos y que contiene toda la información relevante sobre el usuario, el tipo de operación disponible (introducción, cambio o recuperación de contraseña), su fecha de expiración, su lenguaje y país y otros datos que puedan ser necesarios. Por tanto, a nivel de lógica de negocio, la aplicación solo almacenará temporalmente y enviará el *ticket* y aquella información que el usuario introduzca (como su correo electrónico o sus contraseñas).

6.1.1 Módulo Core

Este módulo agrupa a todos aquellos servicios y utilidades cuyo ámbito de uso dentro de la aplicación sea general. Los servicios y utilidades alojados en el módulo Core pueden ser accesibles desde cualquier parte de la aplicación. A continuación se va a indicar aquellas funcionalidades que son responsabilidad de este módulo y que están distribuidas adecuadamente en servicios.

En primer lugar, este módulo debe encargarse de la obtención de traducciones para todos los textos de la aplicación, conveniando con el servidor o con el navegador del usuario el lenguaje más adecuado para las traducciones. Debe obtener también la lista de idiomas disponibles traducida al idioma que se esté mostrando en ese momento.

A nivel de seguridad, se encarga de validar el *ticket*, que recibirán algunas pantallas a través de la ruta, y de almacenarlo y ponerlo a disposición de otros componentes y servicios en el resto de la aplicación.

Continuando el tema de la seguridad, todas las peticiones al servidor serán firmadas utilizando una librería de la empresa. Para conseguirlo, se utiliza un interceptor, que es una clase que intercepta todas las llamadas a los servidores y les añade una cabecera con los parámetros necesarios para una correcta comunicación con el servidor y con el contenido firmado, garantizando que éste no haya sido modificado durante la comunicación.

Por último, a nivel estético, el módulo Core debe cargar los colores, el logo y la imagen corporativas en un servicio y ponerlas a disposición del resto de la aplicación así como cambiar el título y el icono de la pestaña del navegador en cada momento.

6.1.2 Módulo Static

Este módulo contiene el esqueleto de la aplicación, es decir, todos aquellos elementos estáticos que sean compartidos en la mayor parte de las pantallas. Está compuesto por un único componente y un fichero con las rutas hacia el resto de los componentes que forman las pantallas de la aplicación. En los casos necesarios, se han definido *guards* que redirigen al usuario a otras pantallas si no se puede validar el *ticket*.

6.1.3 Módulo Features

En el módulo *Features* se localizan los componentes que forman las distintas pantallas visibles por el usuario y que le permiten llevar a cabo los procesos de introducción, cambio y recuperación de contraseña. También contiene los servicios específicos que permitan llevar a cabo la lógica de cada pantalla.

Este módulo alberga cuatro páginas distintas: cambiar contraseña, nueva contraseña, recuperar contraseña y página de alerta.

Cada una de ellas está formada por un componente que, a partir de otros subcomponentes (explicado más adelante en el módulo Shared), crea la interfaz que permite al usuario llevar a cabo uno de los procesos antes mencionados en cada página. Este componente tiene también como responsabilidad reaccionar a las interacciones del usuario, modificando la interfaz y enviando a los servidores la información necesaria.

Junto con cada uno de los componentes que dan lugar a una página distinta, se ha creado también un servicio que les permite realizar las peticiones al servidor específicas de cada pantalla (ej. Petición al servidor de envío de un correo de recuperación de contraseña).

6.1.4 Módulo Shared

Este módulo almacena todos los componentes que son susceptibles de ser reutilizados en varias pantallas o de ser reutilizados en otras aplicaciones de la empresa, aunque vayan a ser utilizados una única vez. Este módulo no contiene ningún servicio.

Los componentes creados en el módulo Shared han sido diseñados para estar desacoplados de la lógica de la aplicación, facilitando su posterior reutilización en aplicaciones completamente distintas de la empresa.Todos los componentes de este módulo son utilizados en los módulos Static y Features para al creación del las interfaces de usuario para las distintas páginas.

6.2 Esqueleto de la aplicación

En el entorno del desarrollo de interfaces gráficas de usuario, podemos considerar que el esqueleto de una aplicación *web* constituye el conjunto de elementos de la interfaz que son compartidos por la mayoría de pantallas que se van a diseñar.

En el caso de esta aplicación, el esqueleto adoptará dos formas distintas dependiendo del tamaño de la pantalla desde el que se vaya a visualizar. En estas dos formas hay una serie de elementos que son fijos y visibles en la mayor parte de las páginas. Las partes del esqueleto que se marquen como «Contenido» más adelante son las que serán sustituidas por los componentes que formarán cada una de las páginas.

Para pantallas medianas y grandes, como en un ordenador de escritorio o un ordenador portátil, el diseño del esqueleto es el que se puede visualizar en la Figura 3, más adelante. Puede percibirse en este boceto la existencia de dos zonas bien delimitadas: la imagen de fondo, a la izquierda y el contenido, a la derecha. Se ha optado por este diseño, en vez de posicionar el contenido en el centro de la pantalla y a pantalla completa, debido a que el futuro contenido va a ocupar un porcentaje relativamente bajo del área visible, por lo que, por razones estéticas, se va a implementar un diseño de pantalla partida. Ambas partes de la pantalla cambiarán de anchura dependiendo del dispositivo y se ajustarán al espacio disponible.

Figura 3: Boceto del esqueleto de la aplicación web (versión de escritorio)

Para dispositivos con pantallas pequeñas (principalmente teléfonos inteligentes o tabletas pequeñas) en los que el uso del esqueleto anterior no permitiría visualizar correctamente el contenido, se ha optado por el diseño mostrado en la Figura 4. En la imagen se puede ver una sección en la parte superior que contiene el logo de la compañía automovilística, permitiendo al usuario discernir en cual de ellas está realizando el proceso de cambio de contraseña (algunos

usuarios disponen de una cuenta en más de una compañía automovilística). En el caso del esqueleto anterior, el usuario puede percibir la compañía automovilística a partir de la imagen de fondo (que contrendrá un producto reconocible de la compañía) o del logotipo de la compañía en la pestaña de su navegador.

En la mayoría de dispositivos móviles, al orientarse horizontalmente, la aplicación detectará ese cambio de orientación y configurará el esqueleto de la aplicación a pantalla partida, como si de una pantalla mediana o grande se tratase.

Figura 4: Boceto del esqueleto de la aplicación web (versión móvil)

A nivel técnico, ambos esqueletos están implementados en el mismo componente. El cambio de disposición de los elementos se hace exclusivamente a través de estilos SASS.

La colocación del logotipo de la compañía en la versión móvil y de la imagen de fondo en la versión de escritorio es responsabilidad de este componente y, por tanto, ninguno de los componentes «hijo» tendrá responsabilidad de cargarlas.

La sección marcada como «Contenido» en los bocetos será estética y funcionalmente idéntica, tanto en la versión móvil como en la versión de escritorio y, por lo tanto, por brevedad, en los siguientes apartados solo apareceran bocetos e imágenes de la versión de escritorio.

6.3 Páginas

En el contexto de las aplicaciones *web* creadas con Angular, se puede definir una ruta como el conjunto de componentes anidados que dan lugar al contenido visible por el usuario al acceder a una dirección *web* (*URL*). De esta forma, dependiendo de la *URL*, la aplicación carga unos componentes u otros.

En la Figura 5 pueden observarse las posibles combinaciones de rutas y las páginas que la aplicación cargará en cada una de ellas. Cabe destacar que el término «compañia» en la *URL* de la raíz de la aplicación será sustituida por el nombre de cada una de las empresas en las que se use esta aplicación (ej. Audi) y el término «ticket» localizado en las páginas «Cambiar contraseña» y «Nueva contraseña» será sustituido por el código de un ticket (ej. -6739032286081037048).

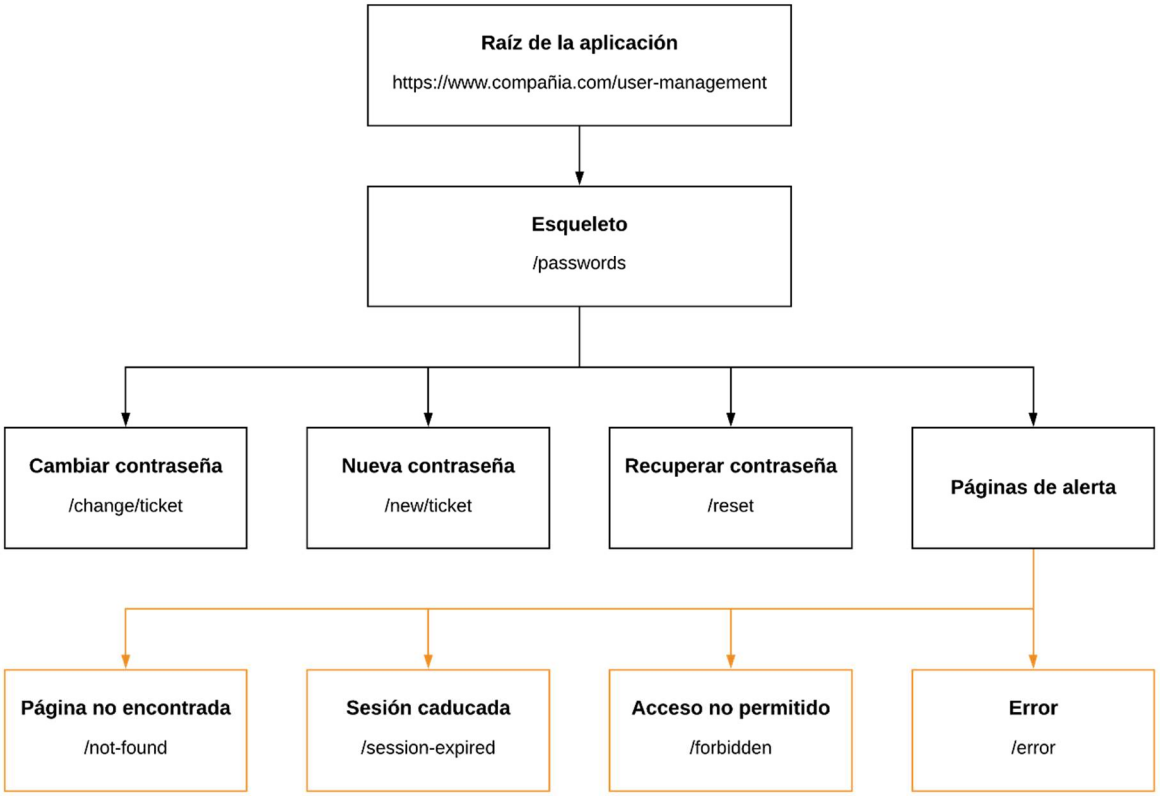

Figura 5: Diagrama de páginas y rutas de la aplicación web

Cada una de las cajas negras del diagrama anterior designa un componente (creado junto a un servicio) de la aplicación. Dentro de cada caja se encuentran dos rótulos: el primer rótulo resaltado en negrita indica el nombre del componente y el segundo rótulo, indica la parte de la *URL* final correspondiente a ese componente. De este modo, para conocer la ruta de un componente, se suman los fragmentos de las URL de todos los componentes que le preceden jerárquicamente.

Por ejemplo, en el caso de la página «Cambiar contraseña», el primer componente que se carga es la propia aplicación (la raíz de la aplicación), seguidamente se carga el esqueleto y por último el componente «Cambiar contraseña». De este modo, la *URL* resultante sería *https://www.compañia.com/user-management/passwords/change/ticket* (*https://www.compañia.com/user-management* + */passwords* + */change/ticket*).

En el tercer nivel del diagrama anterior se encuentra la caja «Páginas de alerta», que da lugar a otro nivel jerárquico, resaltado en naranja. En las situaciones en las que el proceso que debe llevar a cabo el usuario de la aplicación no puede continuar por impedimentos de cualquier tipo, se redirige a éste a una página de alerta. Sin embargo, dado que el formato de todas estas páginas de alerta es el mismo, no es viable crear un componente específico para cada una de estas rutas (como sí ocurre en las páginas de cambiar, nueva y recuperar contraseña). Por ello, se ha implementado en un único componente que modifica la alerta visualizada (en naranja) en función de un argumento proporcionado en el archivo de rutas (explicado en el apartado anterior Módulo Shared).

6.3.1 Cambiar contraseña

La página «Cambiar contraseña» tiene como principal función permitir al usuario realizar el cambio de su contraseña, introduciendo su contraseña antigua, su contraseña nueva y confirmando la contraseña nueva. Esta página resulta conveniente en las situaciones en las que el usuario, por motivos de seguridad, desee cambiar su contraseña y conozca su contraseña antigua, evitando el acceso al correo electrónico y agilizando el proceso.

El usuario accede a esta página redirigido desde la plataforma de gestión de la compañía correspondiente. Cada plataforma tiene la responsabilidad de generar un enlace que contenga el *ticket* con todos los datos del usuario y del proceso.

Cuando el usuario accede a la aplicación a través del enlace mencionado, se carga la aplicación, el esqueleto y, antes de cargarse la página «Cambiar contraseña», un *guard* intercepta el proceso de carga y envía una petición al servidor para comprobar la validez del *ticket*. Si el *ticket* es válido, prosigue la carga de la página «Cambiar contraseña». Si no es válido, se redirige al usuario a la página de alerta «Acceso no permitido».

En la Figura 6 puede encontrarse un boceto de la interfaz de esta página. En el lado derecho de la imagen, en orden descendente puede observarse:

- Botón con un icono de la bandera del país al que pertenece el lenguaje y la variante idiomática de los textos visibles. Al ser pulsado da acceso a un menú de cambio de lenguaje, ilustrado en la Figura 7. Todas las banderas de la Figura 7 son botones que, al ser pulsados, cargan las traducciones del idioma seleccionado.
- Título de la página. Aclara al usuario el proceso o gestión que va a realizar.
- Campo «Contraseña anterior» del formulario. En este campo, el usuario debe introducir la contraseña actual de la cuenta.
- Campo «Contraseña nueva» del formulario. En este campo, el usuario debe introducir la contraseña que desea establecer para su cuenta.

- Campo «Confirmar contraseña» del formulario. En este campo, el usuario debe volver a introducir la contraseña que desea establecer para su cuenta.
- Cuadro de reglas. Durante el proceso de carga de la página de cambio de contraseña se realiza una petición al servidor para obtener las reglas de creación de contraseñas que el usuario debe cumplir. Tras la recepción de las reglas, éstas son mostradas en una lista en el cuadro de reglas, debidamente traducidas.
- Botón de cambio de contraseña. Permanece bloqueado a la pulsación mientras todos los campos del formulario no estén rellenados y los campos «Contraseña nueva» y «Confirmar contraseña» no coincidan. Cuando se cumplan las condiciones anteriores, el botón cambiará a nivel estético y estará disponible para su pulsación.

Cuando el usuario pulse el botón de cambio de contraseña, a través del servicio creado junto a este componente, se realiza una petición al servidor para realizar el cambio de contraseña. Si se cumplen todas las reglas, el usuario visualizará en pantalla un mensaje de éxito junto con un botón que le permita volver a la plataforma de gestión, como puede verse en la Figura 8. En caso de no cumplirse alguna regla, el servidor devolverá la clave de traducción de la regla incumplida, que automáticamente se traduce al idioma seleccionado. Esta regla incumplida será visualizada en el cuadro de reglas y se resaltarán los campos del formulario en color rojo, para ayudar al usuario a identificar el problema. Si el usuario no puede completar el proceso debido a razones técnicas, se le redirigirá a una pantalla de alerta.

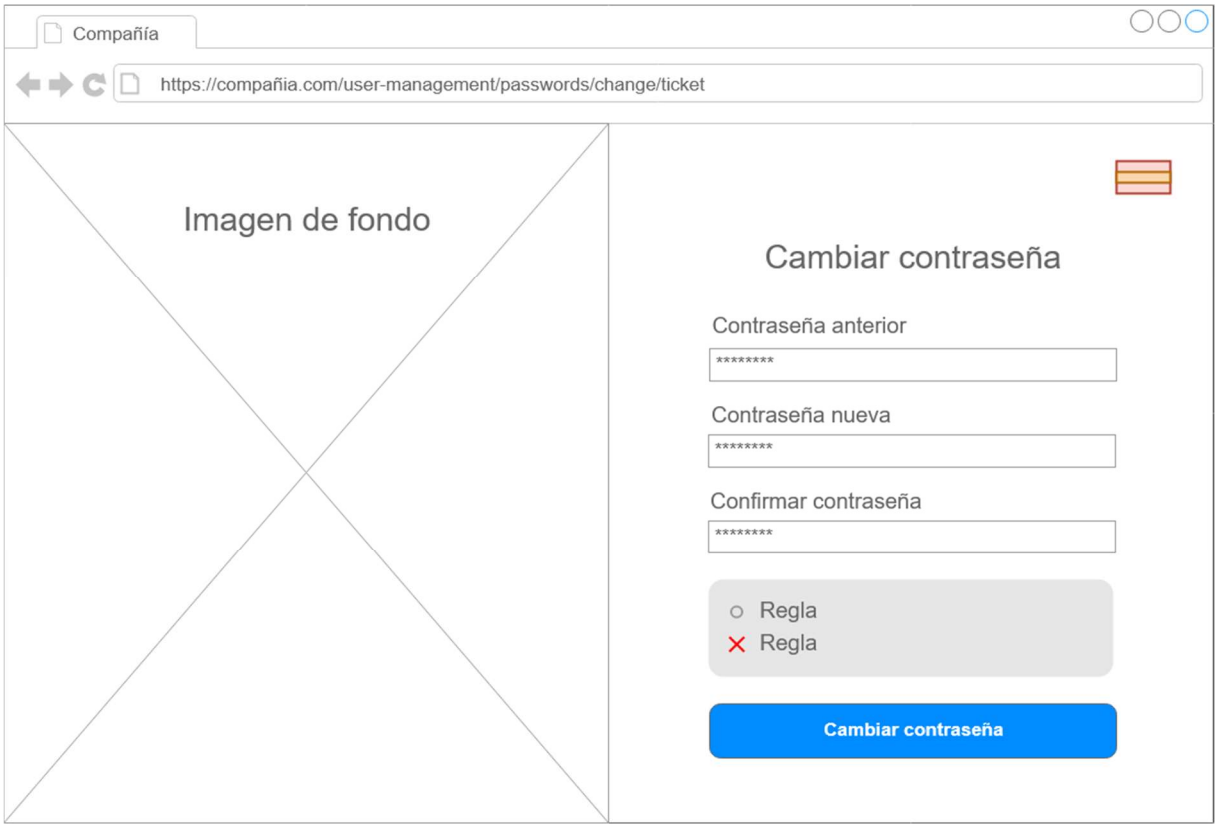

Figura 6: Boceto de la página de cambio de contraseña de la aplicación web

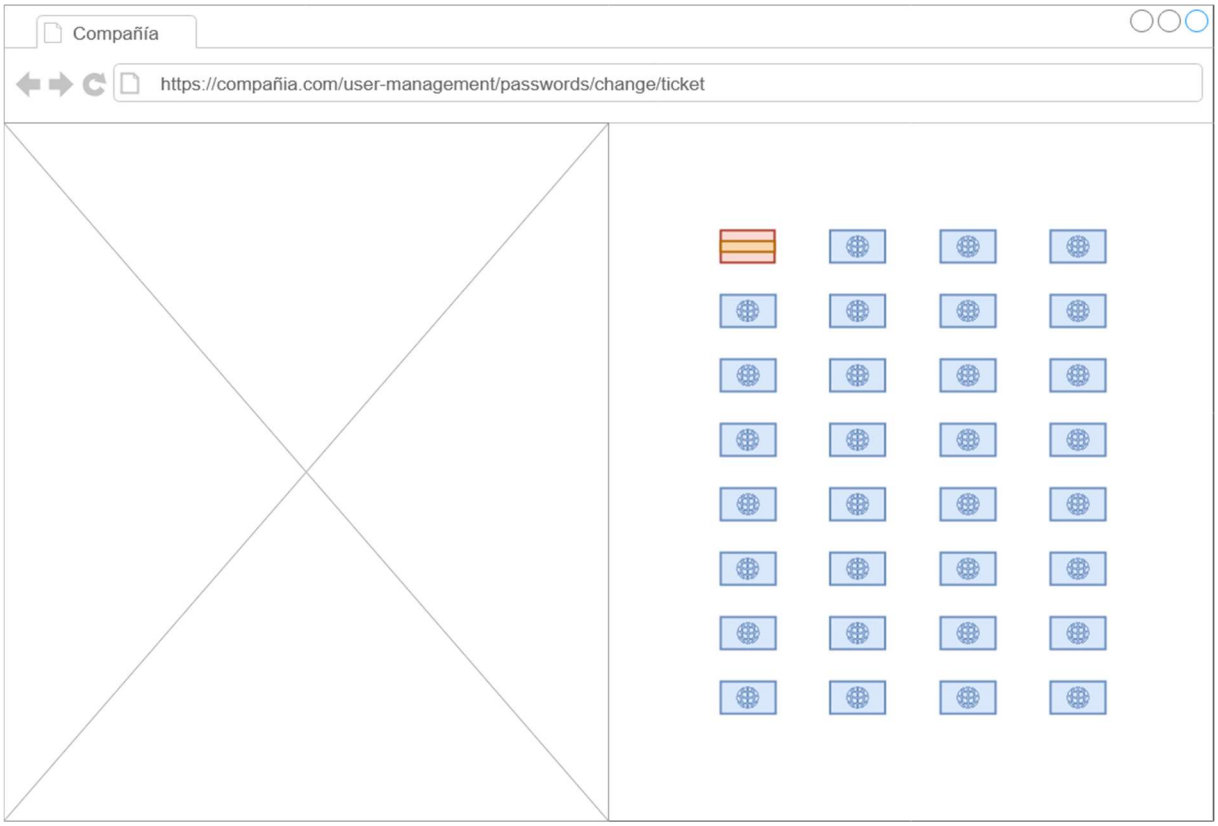

Figura 7: Boceto del menú de cambio de idioma de la aplicación web

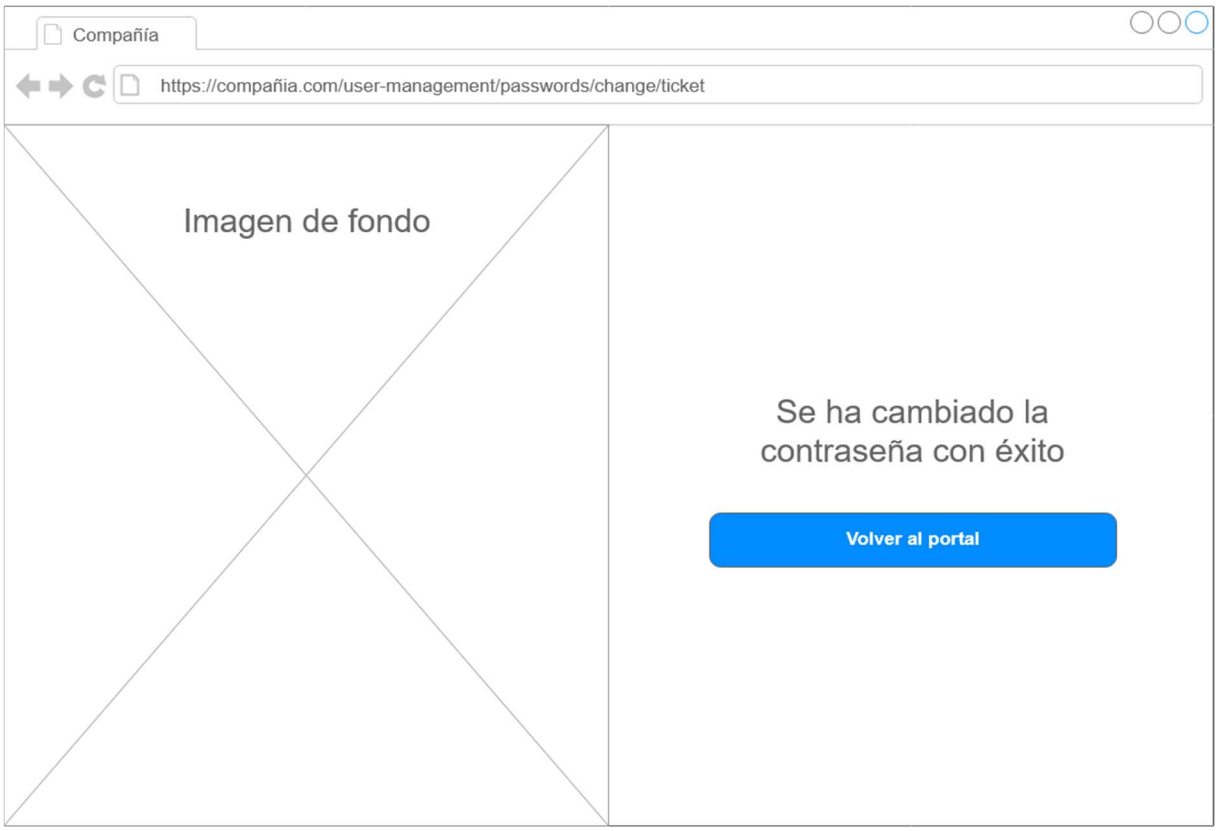

Figura 8: Boceto de la pantalla de éxito de la aplicación web

Para facilitar la lectura de este documento, de ahora en adelante, los bocetos e imágenes de las versión móvil de la aplicación *web* se mostrarán en el Anexo B.

6.3.2 Nueva contraseña

La página «Nueva contraseña» permite al usuario establecer una nueva contraseña para su cuenta. Esta página abarca la funcionalidad de tres casos descritos en el apartado 2.3: introducción de la primera contraseña por parte de nuevos usuarios, introducción de una nueva contraseña a usuarios a los que los administradores les han forzado a cambiar la contraseña por motivos de seguridad y a aquellos usuarios que han pedido un correo electrónico de recuperación de contraseña.

En todos los casos anteriormente mencionados, el usuario accede a esta página a través de un enlace enviado por correo electrónico y que contiene un *ticket* personalizado. En el primer y segundo caso, el envío del correo electrónico es realizado por la plataforma de gestión y, por tanto, su uso y su diseño no se abordarán en este documento. El envío del correo electrónico de recuperación de contraseña sí se abordará más tarde en el apartado 6.3.3.

A nivel de interfaz y a nivel funcional, como puede observarse en la Figura 9, esta pantalla es muy similar a la descrita en el apartado anterior. Se diferencian principalmente en la ausencia del campo «Contraseña anterior» y en los textos visualizados.

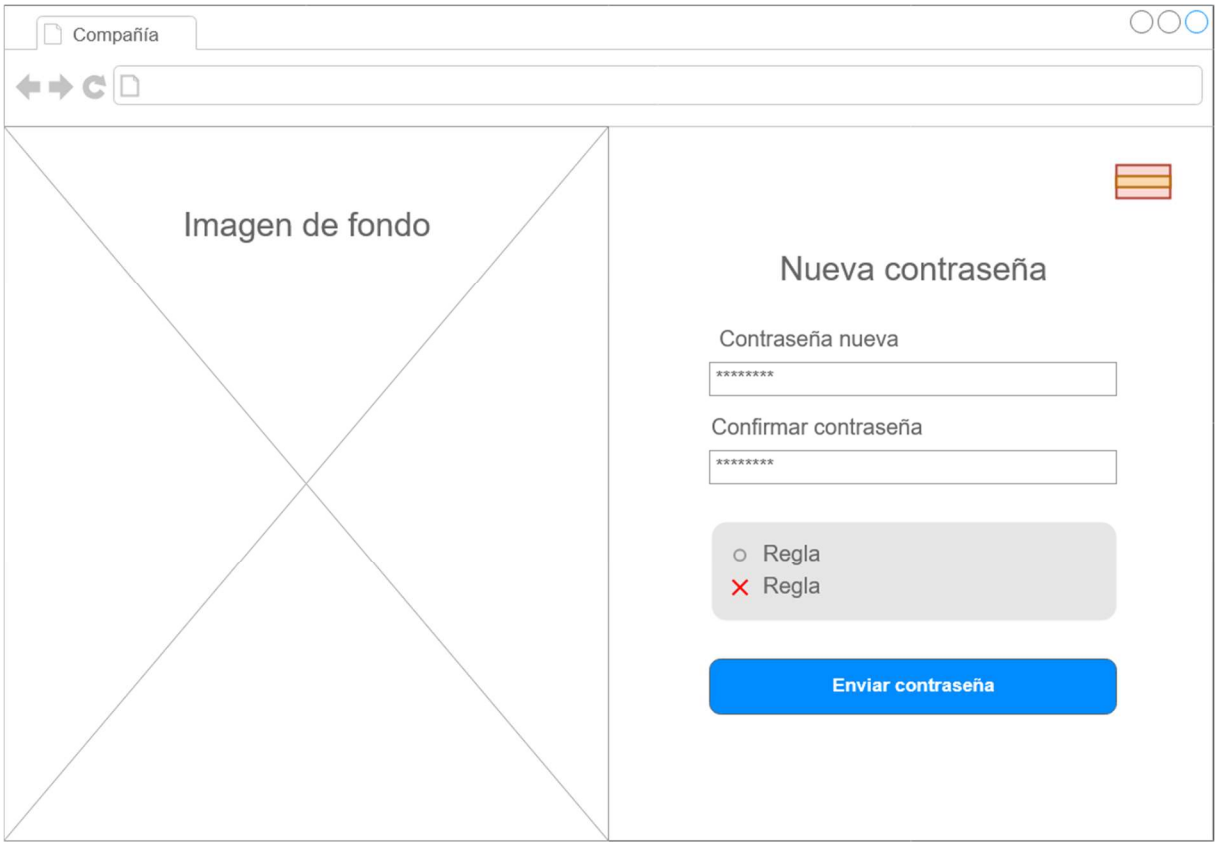

Figura 9: Boceto de la página de nueva contraseña de la aplicación web

6.3.3 Recuperar contraseña

La página «Recuperar contraseña» permite al usuario realizar una petición para recibir un correo electrónico con instrucciones para recuperar su contraseña. Es útil cuando el usuario ha olvidado su contraseña y necesita introducir una nueva para poder seguir haciendo uso de su cuenta en la plataforma de gestión.

A esta página se accede a partir de un enlace (Con etiqueta *Forgot password?*, traducido como ¿Contraseña olvidada?) localizado en el portal de inicio de sesión de la plataforma de gestión. A diferencia de los casos de cambio y nueva contraseña, el enlace no contiene ningún *ticket*, dado que es imposible conocer los datos del usuario si éste no ha iniciado sesión. El hecho de no recibir *ticket* en esta página implica que ya no es necesario el uso de un *guard* para proteger esta ruta.

El boceto de esta página puede contemplarse en la Figura 10, observando, en el lado derecho y en orden descendente, los siguientes elementos:

- Botón de cambio de lenguaje.
- Título de la página.
- Selector de campo. Consiste en un grupo de dos o más botones en el que solo uno de ellos puede estar seleccionado (resaltado en un color corporativo diferente al gris). Para poder recibir un enlace con instrucciones para la recuperación de la contraseña del usuario, éste debe especificar algún elemento identificativo de su cuenta. En esta aplicación, por el momento, se ofrecen dos opciones: envío directo a la dirección de correo electrónico ofrecida o envío a la dirección de correo electrónico vinculada al nombre de usuario proporcionado. Al cargar la página aparece activo el primer botón, aunque el usuario puede activar el segundo si así lo desea.
- Cuadro de instrucciones. Contiene aclaraciones e instrucciones convenientes para el usuario relacionadas con el envío del correo electrónico.
- Botón de envío de instrucciones de recuperación.

El botón de cambio de lenguaje funciona de igual manera que en casos anteriores, desplegando el menú de cambio de lenguaje. El botón de envío de instrucciones de recuperación permanece bloqueado hasta que el usuario introduzca su dirección de correo electrónico o su usuario. Una vez que el usuario pulse dicho botón, dependiendo de la opción elegida en el selector de campo se llamará a un método o a otro del servicio creado junto a este componente. En ambos casos, se mostrará un mensaje de éxito al usuario, tanto si existe como si no, por motivos de seguridad. Si el servidor no respondiese a las peticiones, sí se redirigía a una página de alerta, informando al usuario de que el proceso no puede continuar por motivos ajenos al mismo.

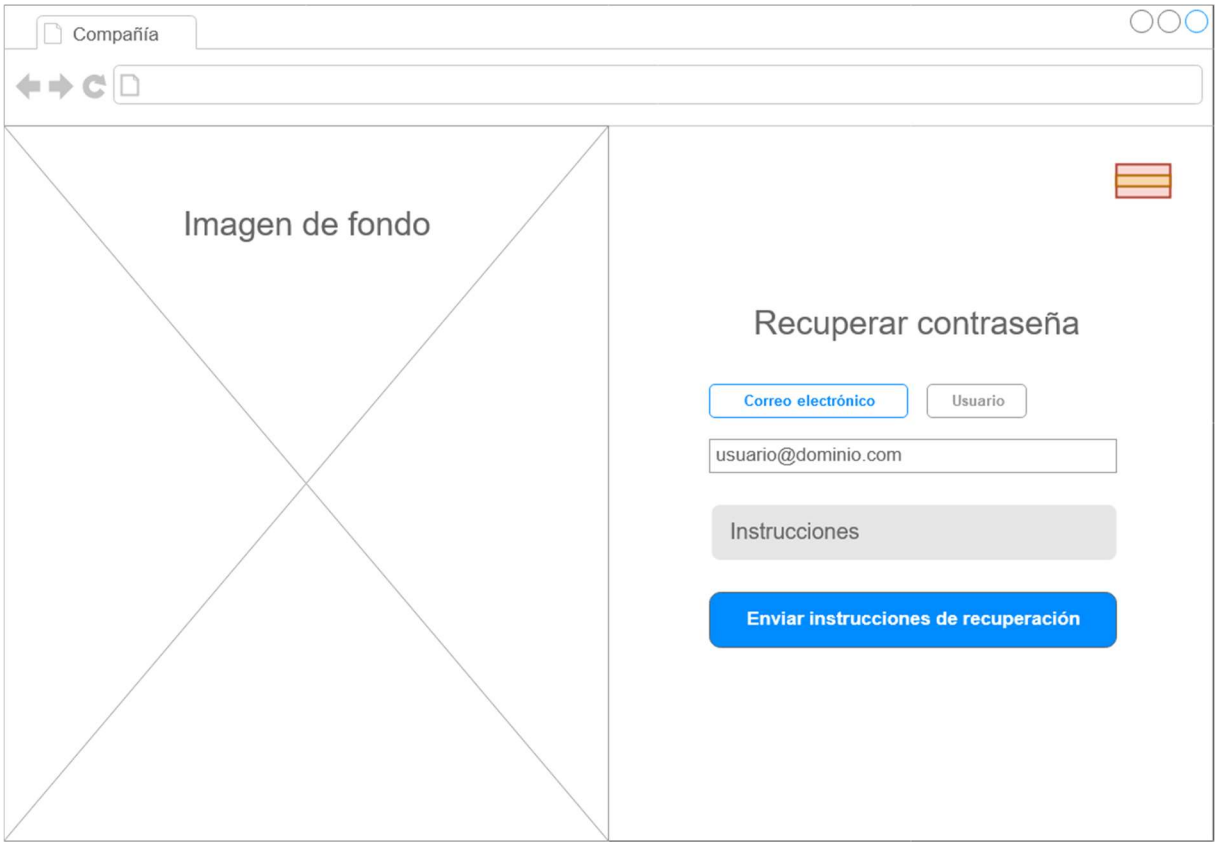

Figura 10: Boceto de la página de recuperar contraseña de la aplicación web

6.3.4 Páginas de alerta

Cuando un usuario normal utiliza una aplicación, éste realiza un conjunto de acciones en un entorno diseñado por el desarrollador y en una secuencia establecida. Un ejemplo de esto sería el cambio de contraseña, una experiencia en la que el usuario comienza introduciendo los datos que se le indican en los campos del formulario para, a continuación enviarlo al servidor y esperar una respuesta de éste, indicando que el proceso ha culminado con éxito. Sin embargo, en el mundo real no siempre sucede todo en el orden en el que ha sido diseñado o, directamente, los usuarios no siempre pueden acabar el proceso o gestión que han comenzado.

Es por ello, que urge diseñar páginas que ilustren y hagan entender al usuario qué problemas han podido suceder durante el uso de la aplicación, de forma que éstos puedan reaccionar convenientemente.

Existen un conjunto de problemas técnicos y de seguridad que debe enfrentar esta aplicación y para los que se ha diseñado una página, como se ha indicado en la Figura 5:

- Página no encontrada. El usuario ha entrado accidental o voluntariamente a una página dentro de este dominio que no existe.
- Sesión caducada. El usuario ha accedido a un enlace que, por motivos de seguridad, ha caducado y ya no es válido.

- Acceso no permitido. El sistema ha generado un enlace con un *ticket* erroneo o el usuario ha manipulado el enlace.
- Error. La aplicación ha sido servida al usuario pero ésta no ha podido conectar con las APIs de la empresa y, por tanto, no puede llevarse a cabo ninguna operación.

Durante todo el apartado 6.3 se ha seguido el esquema de componente y servicio por página. Dicho componente es cargado una única vez en el archivo de rutas. Sin embargo, para evitar la repetición de código, se ha creado un solo componente para todas las alertas. Para que el componente pueda conocer qué alerta mostrar en pantalla en cada ocasión, se utiliza varias veces en el archivo de rutas y en cada una de ellas se le pasa un argumento indicándole que alerta es para cada ruta.

En la Figura 11 puede observarse la estructura de la información que contendrá la página de alerta. Consta de un título breve e indicativo de la alerta (ej. Sesión caducada), una descripción más detallada del problema y de sus posibles soluciones y, por último, un botón para volver a la plataforma de gestión.

Cabe destacar que los textos de las alertas son traducidos el idioma del usuario (si se dispone de información del *ticket*) o del primer idioma del navegador que coincida con los idiomas disponibles en el servidor.

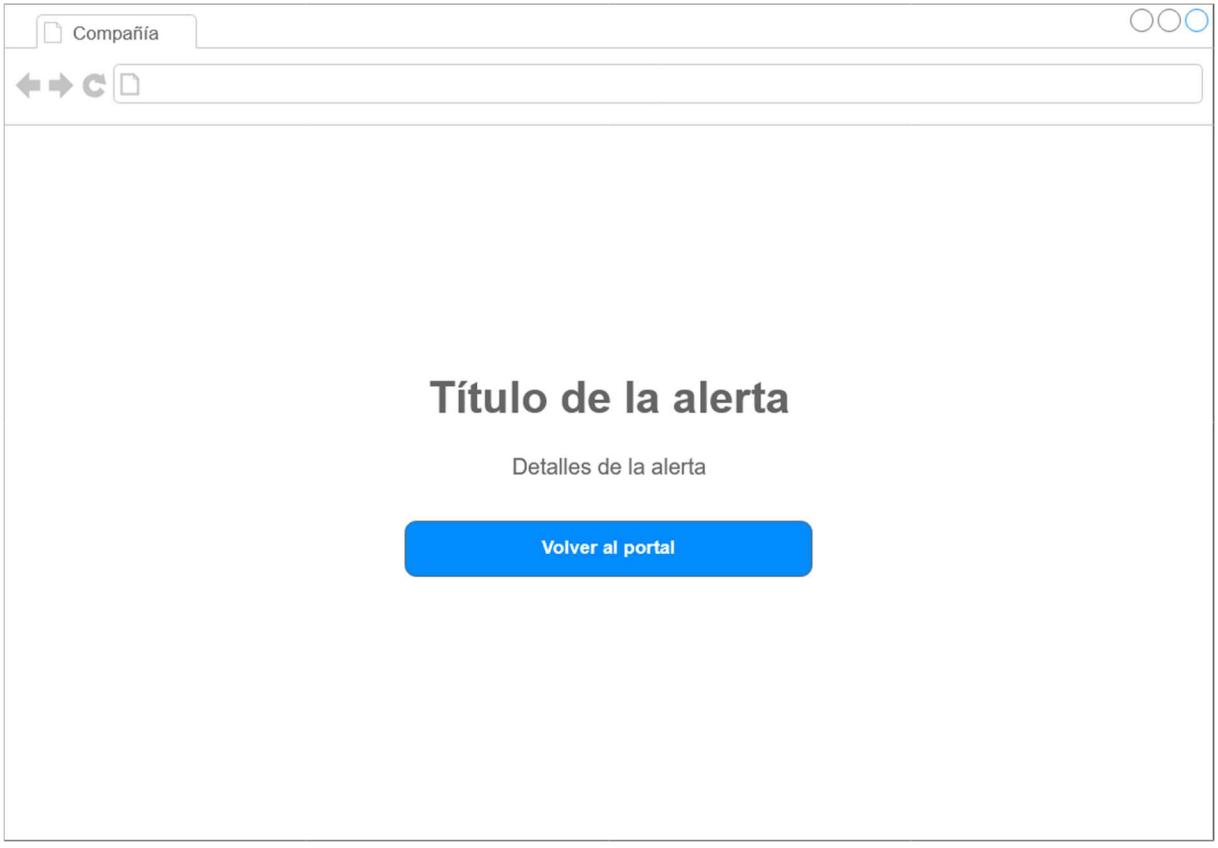

Figura 11: Boceto de la página de alerta de la aplicación web

7
Diseño de las adaptaciones al servidor

Con el objetivo de comprender más facilmente las modificaciones que van a diseñarse y adesarrollarse, es útil conocer, en líneas generales, cómo está estructurado el código en el *backend* de la empresa.

Los servicios que abastecen de datos a los productos de la empresa, ODEC, que proporciona a sus clientes, están organizados y expuestos públicamente en programas desarrollados en Java y siguiendo la arquitectura *REST*.

REST (*REpresentational State Transfer*) es un estilo de arquitectura para la creación de servicios *web* [9]. Entre sus ventajas se encuentran su independencia del lenguaje de programación y de la plataforma, su escalabilidad y fiabilidad y su facilidad para separar conceptualmente las funcionalidades del servidor del propio almacenamiento de la información.

Para que pueda decirse que un programa cumple con la arquitectura *REST*, éste debe poseer las siguientes características:

- Uso del protocolo HTTP y de sus tipos de operaciones: POST (crear), GET (obtener información), PUT (editar) y DELETE (eliminar).
- Ausencia de estado. Todas las comunicaciones deben contener toda la información necesaria para llevar a cabo el proceso requerido por la petición y, por tanto, el servidor no debe almacenar ninguna información del cliente.
- Uso de la URI (Uniform Resource Identifier) para identificar cada recurso de forma única.
- Sistema de capas. Cada uno de los pasos del proceso, como la comunicación, el tratamiento y el almacenamiento de la información, deben estar distribuidos en capas, aislándolos y estableciendo una estructura clara para el manejo de la información.
- Uso de hipermedios. Las respuestas del servidor deben ofrecer al cliente enlaces con los que realizar acciones concretas sobre la información recibida en la respuesta.

De este modo, para realizar una operación, lo que necesita el cliente es definir qué tipo de operación desea hacer (ej. *GET*), a dónde (la *URI* concreta), proveer los argumentos (la información) requeridos por el servidor y reaccionar a las distintas respuestas que éste pueda devolver.

Siguiendo con la arquitectura general del *backend* de la empresa, todos los servicios mencionados anteriormente no están desarrollados en un gran programa monolítico, sino en un grupo de *APIs* discriminadas por conceptos y creadas por grupos de desarrolladores separados, atendiendo a las necesidades de la empresa en cada momento. Esta relación puede percibirse más claramente en la

Figura 12, a continuación.

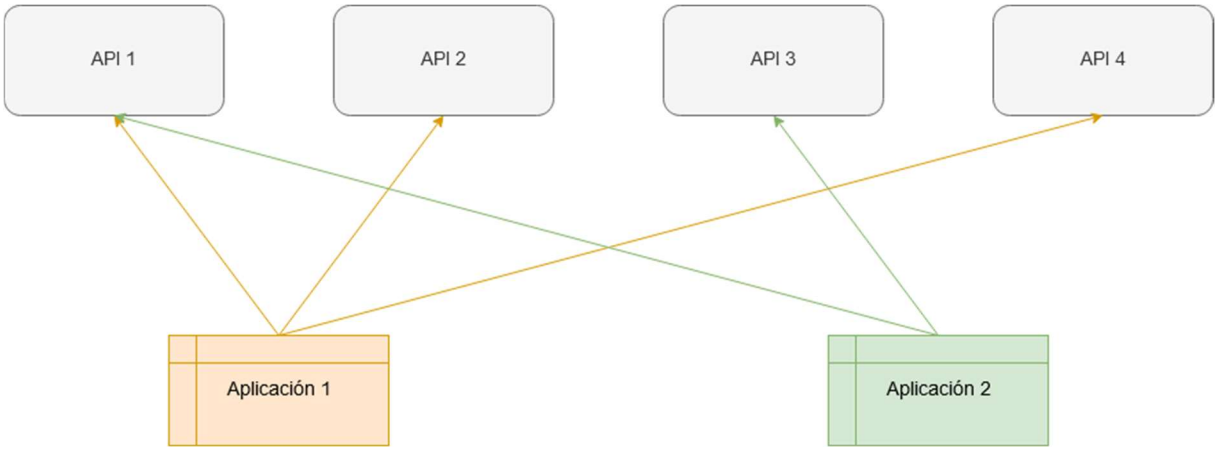

Figura 12: Diagrama de peticiones de varias aplicaciones a un grupo de APIs

La aplicación *web* diseñada en el capítulo anterior requiere hacer operaciones y obtener información de dos bloques conceptuales bien distintos: las cuentas de los usuarios de las plataformas de gestión y las traducciones de los textos que se muestran en la aplicación.

El primer bloque se lleva a cabo en una *API* conocida en la empresa como la *API* de usuarios (*users API*). El segundo bloque se lleva a cabo en otra *API*, la *API* de mensajes (*messages API*).

En ambos casos se sigue una estructura con varias capas, aunque solo dos de ellas van a ser adaptadas, la capa *REST* y la capa *EJB*.

La capa *REST* contiene los métodos específicos para le gestión de las solicitudes *HTTP* y las respuestas del servidor a las mismas.

La capa *EJB* (*Enterprise JavaBeans*) tiene acceso a los recursos de almacenamiento del servidor, y tiene como función el procesamiento de la información y su almacenamiento.

A nivel de seguridad, la API de usuario está protegida, entre otras técnicas no abarcadas en este documento, de tres formas distintas:

- Las operaciones requieren de un *ticket* generado específicamente por cada usuario y operación, con tipos de operación específicos y fecha de caducidad.
- Todas las operaciones que se realizan con un *ticket* son firmadas en las cabeceras de las peticiones con código de autentifiación de mensajes en clave-hash (HMAC).
- Conexiones realizadas bajo el protocolo HTTPS, mucho más seguro que el protocolo HTTP estándar, aún utilizado en muchos sitios *web*.

El *ticket* contiene información relevante sobre el usuario y es indispensable para el funcionamiento de los métodos que lo emplean. A partir del *ticket* puede obtenerse el identificador único del usuario, el tipo de operación (introducción, cambio o recuperación de contraseña), su fecha de caducidad y si el *ticket* está cerrado, es decir, ha sido usado o ha caducado. Un *ticket* típico en

este servidor está constituído por un conjunto de números y un guión en algunos casos (ej. -6739032286081037048).

7.1 Capa REST

Esta capa actúa exclusivamente en todo lo relacionado con las comunicaciones a través del protocolo *HTTP*, resolviendo el método u operación (*POST, GET, PUT, DELETE*), la ruta (*URI*) y los argumentos. Con ellos invoca a la capa *EJB*, le proporciona un conjunto de argumentos y espera hasta recibir los datos requeridos para la respuesta o una excepción si no se han podido obtener. Acto seguido, genera una respuesta en formato *JSON* y devuelve un código HTTP indicando al usuario el estado (*status*) de la respuesta.

El estado (*status*) en una comunicación por el protocolo HTTP es el código que devuelve el servidor junto a su respuesta y que informa al cliente que ha realizado la petición sobre el éxito, o no de ésta y los motivos en el caso de no hacer la operación con éxito. Los códigos utilizados en los métodos diseñados en estas adaptaciones, sus nombres y sus casos de uso pueden observarse en la Tabla 1.

CÓDIGO	NOMBRE	CASO DE USO
200	OK	La petición se ha procesado correctamente.
400	Bad Request	El servidor no procesará la petición debido a un fallo en el formato o el contenido de la misma.
401	Unauthorized	No ha sido posible autenticar el usuario de la petición.
403	Forbidden	El servidor ha autenticado al usuario de la petición pero éste no tiene los privilegios suficientes.
404	Not Found	Indica que la página o el recurso buscados no existen en el servidor.
500	Internal Server Error	Ha sucedido un error dentro del servidor durante el procesamiento de la petición.

Tabla 1: Códigos HTTP de respuesta utilizados en este proyecto

En la Tabla 1 hay tres tipos de códigos dependiendo del primer dígito de éstos:

- Comenzados en 2. Indican que la petición se ha ejecutado correctamente y se devolverá aquello que se ha solicitado (si lo hubiera).
- Comenzados en 4. Indican que la petición no ha podido ser completada debido a errores de la solicitud, bien en el formato o bien en aquello que se pide.
- Comenzados en 5. Indican un fallo interno del servidor, ajeno al cliente o aplicación *web* que está realizando la petición.

Es importante destacar que, en la implementación anterior de la aplicación *web*, no están expuestos públicamente en la capa *REST* algunos de los servicios ya implementados en la capa *EJB*. Por tanto, es necesario la publicación de los métodos *EJB* existentes y de aquellos que van a ser creados.

A continuación, se incluye una lista de fichas con los detalles de los métodos diseñados para disponer públicamente todas las funcionalidades necesarias para la gestión de las contraseñas de las cuentas de los usuarios de la aplicación *web*.

NOMBRE	findOpenTicket
FUNCIÓN	Obtener información del usuario vinculada al ticket y comprobar la existencia y validez del ticket.
ARGUMENTOS	Ticket
RESPUESTA	Información vinculada al ticket
CÓDIGOS	200 – Éxito 401 – Ticket cerrado 404 – Ticket no encontrado 500 – Error del sistema

Tabla 2: Método findOpenTicket (REST)

NOMBRE	changePasswordFromTicket
FUNCIÓN	Sobreescribir la contraseña existente
ARGUMENTOS	Ticket, Contraseña nueva
RESPUESTA	Clave de traducción de regla incumplida de la contraseña (en código 400)

CÓDIGOS	200 – Éxito 400 – La contraseña incumple alguna regla 401 – Ticket cerrado 403 – El ticket no está autorizado a hacer esta operación 404 – Ticket no encontrado 500 – Error del sistema

Tabla 3: Método changePasswordFromTicket (REST)

NOMBRE	replacePasswordFromTicket
FUNCIÓN	Verificar la contraseña antigua y cambiarla por una nueva
ARGUMENTOS	Ticket, Contraseña antigua, Contraseña nueva
RESPUESTA	Clave de traducción de regla incumplida de la contraseña (en código 400)
CÓDIGOS	200 – Éxito 400 – La contraseña incumple alguna regla 401 – Ticket cerrado 403 – El ticket no está autorizado a hacer esta operación 404 – Ticket no encontrado 500 – Error del sistema

Tabla 4: Método replacePasswordFromTicket (REST)

NOMBRE	getRulesFromListWithTicket
FUNCIÓN	Obtener la lista traducida de reglas de creación de contraseñas
ARGUMENTOS	Ticket, (Lenguaje), (Variante)
RESPUESTA	Lista de reglas
CÓDIGOS	200 – Éxito 401 – Ticket cerrado 404 – Ticket no encontrado 500 – Error del sistema

Tabla 5: Método getRulesFromListWithTicket (REST)

En la Tabla 5 pueden encontrarse dos argumentos escritos entre paréntesis. Esto significa que dichos argumentos son opcionales, pues el lenguaje y la variante idiomática preferidos por el

usuario son extraídos en primera instancia del contenido del *ticket*, aunque si el usuario ha seleccionado otro lenguaje a través del menú de cambio de lenguaje de la aplicación *web*, se permite sobeescribir los valores extraídos.

7.2 Capa EJB

La capa EJB usa los argumentos recibidos de la capa *REST*, invoca los métodos del *backend* para el acceso a la base de datos o a métodos de otros *EJB*. Estos métodos generan el modelo de datos de la respuesta y/o lanzan una excepción en las situaciones que les impidan realizar con normalidad la tarea o funcionalidad para la que han sido diseñadas.

Algunos de los métodos que van a describirse a lo largo de este apartado ya han sido creados por otros desarrolladores, aunque han sido adaptados para su funcionamiento con el sistema de *tickets* que se emplea en este proyecto. Otros, sin embargo, necesitan ser implementados íntegramente. En cualquiera de los casos, tanto en el apartado de diseño como el de desarrollo serán pormenorizados de la misma forma.

A continuación se detallan los métodos de la capa *EJB* diseñados para proveer la información requerida por la capa *REST*.

NOMBRE	findOpenTicket
FUNCIÓN	Buscar y extraer de la base de datos un ticket abierto y válido
ARGUMENTOS	Ticket
DEVUELVE	Información vinculada al ticket
EXCEPCIONES	ClosedUserRequestException – El ticket ha sido utilizado o ha caducado. UnknownUserRequestException – El ticket no existe.

Tabla 6: Método findOpenTicket (EJB)

NOMBRE	changePassword
FUNCIÓN	Sobreescribir la contraseña existente del usuario con una nueva
ARGUMENTOS	Ticket, Contraseña nueva
DEVUELVE	Nada
EXCEPCIONES	ClosedUserRequestException – El ticket ha sido utilizado o ha caducado. UnknownUserRequestException – El ticket no existe. UserPasswordException – La contraseña nueva incumple alguna regla.

Tabla 7: Método changePassword (EJB)

NOMBRE	replacePassword
FUNCIÓN	Verificar la contraseña anterior y cambiarla por la nueva
ARGUMENTOS	Ticket, Contraseña anterior, Contraseña nueva
RESPUESTA	Nada
EXCEPCIONES	ClosedUserRequestException – El ticket ha sido utilizado o ha caducado. UnknownUserRequestException – El ticket no existe. UserPasswordException – La contraseña nueva no incumple alguna regla InvalidCurrentPasswordException – La contraseña anterior no es correcta

Tabla 8: Método replacePassword (EJB)

NOMBRE	getRulesPasswordToUserList
FUNCIÓN	Obtener la lista traducida de reglas de creación de contraseñas
ARGUMENTOS	Ticket, (Lenguaje), (Variante)
RESPUESTA	Lista de reglas
EXCEPCIONES	ClosedUserRequestException – El ticket ha sido utilizado o ha caducado. UnknownUserRequestException – El ticket no existe.

Tabla 9: Método getRulesPasswordToUserList (EJB)

Cabe destacar que en los métodos «changePassword» y «replacePassword» la devolución de las reglas a la capa REST se hace mediante un servicio implementado en ese proyecto que cumple la función de mensajero, por lo que éstas no son devueltas directamente por la función cuando es invocada.

8
Desarrollo de la aplicación web

En este apartado se da por hecho que, si el lector necesita utilizar alguna de las herramientas o tecnologías que conforman este proyecto, haya hecho por sí mismo la instalación de las mismas, siguiendo los pasos indicados en sus respectivas *webs* oficiales. En ningún caso se han seguido pasos adicionales a aquellos indicados por los tutoriales oficiales de cada una de las herramientas y tecnologías.

Una vez diseñada la aplicación, el primer paso a tomar es el de la creación de un repositorio en GitLab, donde se subirá el código de la aplicación. Este paso es necesario hacerlo a través de la *web* donde están alojados los repositorios privados de la empresa.

Después de crear el repositorio, se procede a la creación del proyecto utilizando el paquete Angular-CLI, instalado con npm. Sin embargo, antes de esto, conviene visualizar primero en la Tabla 10 aquellos comandos que van a ser utilizados a lo largo del desarrollo de la aplicación.

COMANDO	FUNCIÓN
ng new	Crea un nuevo directorio con la estructura de un proyecto Angular.
ng serve	Compila el código de la aplicación y lo ejecuta.
ng generate	Genera un componente, servicio, guard o módulo, entre otros, a partir de una plantilla.

Tabla 10: Listado de comandos de Angular-CLI utilizados en esta aplicación

Utilizando el comando «ng new user-management» en la consola de comandos (CMD) o bien en la consola integrada de Visual Studio Code, en el directorio donde se vaya a realizar el desarrollo de la aplicación, se crea el proyecto inicial de la misma. Cabe aclarar que «user-management» es el nombre con el que se identifica el proyecto en la empresa, ODEC.

Acto seguido, se ejecuta el comando «ng serve» para arrancar la aplicación en la dirección y puerto por defecto (http://localhost:4200). Se accede a la dirección con un navegador, en este caso Firefox y se comprueba que el proyecto funciona y ha sido creado correctamente.

Una vez que se ha comprobado que el proyecto ha sido generado adecuadamente, se sube a GitLab. Para facilitar la subida del código, en este trabajo se ha utilizado la funcionalidad de control de código fuente de Visual Studio Code, que permite más facilmente separar las modificaciones hechas en cada jornada en varios envíos (*commits*) diferentes.

8.1 Arquitectura de la aplicación

Cuando se crea un proyecto en Angular-CLI, se genera una plantilla con la estructura mostrada en la Figura 13.

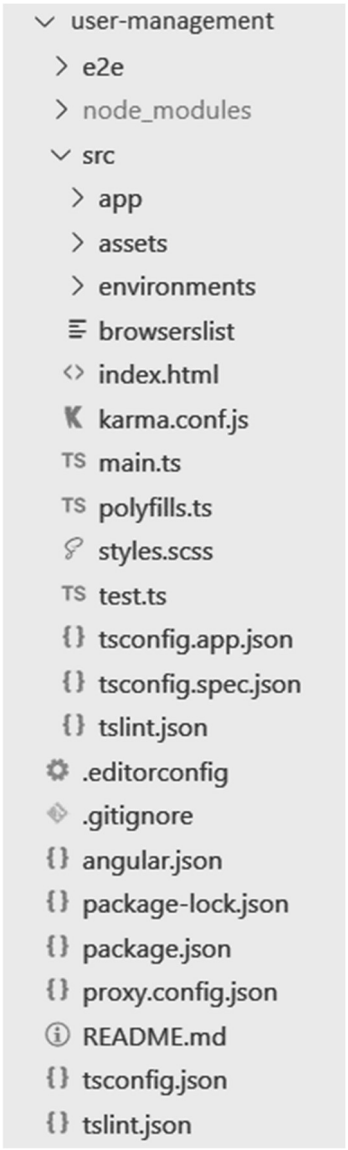

Figura 13: Estructura inicial de la aplicación web

En esta plantilla hay una gran cantidad de archivos, pero aquí se tratarán aquellos cuyas funcionalidades afecten directamente al desarrollo de la aplicación.

En el primer nivel de la jerarquía, los archivos más importantes son:

- angular.json: Contiene las configuraciones generales del proyecto.
- package.json: Contiene la lista de paquetes instalados a través de npm y los *scripts* guardados para su uso en el proyecto.
- proxy.config.json: Contiene configuraciones que permitan al proyecto conectarse a los servidores de la empresa desde fuera de sus redes internas (con VPN).

La carpeta «src» contiene el código de la aplicación que va a ser compilado y ejecutado posteriormente. Dentro de ésta destacan las siguientes carpetas:

- app: Contiene los módulos, componentes, servicios y guards.
- assets: Contiene las imágenes, colores y logos corporativos.

Es en la carpeta «app» donde se crean el primer componente, el archivo de rutas y los cuatro grandes módulos diseñados previamente en el capítulo 7. Desde el directorio «app» y con el comando «ng generate module *nombre_del_modulo*» se crearán los módulos «Core», «Static», «Features» y «Shared».

En la carpeta «assets» se almacenan una imagen y un logo para cada una de las marcas con las que se va a implementar este proyecto inicialmente, Audi y Volkswagen. También contiene una carpeta llamada «flags», con archivos .svg de las banderas de los paises que se utilizarán para el menú de cambio de lenguaje. Por último, se ha creado el archivo «themes.ts» que

contiene la información específica de cada marca, como el título de la página, el enlace para volver a la plataforma, las rutas de las imágenes y los colores corporativos definidos para cada proyecto, todo ello ilustrado más claramente en la Figura 14.

```
export interface Theme {
  pageTitle: string,
  returnURL: string,
  images: {
    logo: string,
    background: string
  }
  colors: {
    accent: string,
    accentLight: string,
    contrastDark: string,
    contrastLight: string
  };
}

const AUDI: Theme = {
  pageTitle: "CEM Audi",
  returnURL: "https://www.cem-audi.com",
  images: {
    logo: "assets/images/audi_logo.png",
    background: "assets/images/audi_main.jpg"
  },
  colors: {
    accent: "#BB0A30",
    accentLight: "#C12042",
    contrastDark: "#575756",
    contrastLight: "#E6E6E6"
  }
};
```

Figura 14: Temas de la aplicación web

En la figura anterior puede observarse que se ha creado una interfaz para forzar una estructura común en todos los temas, evitando que otro desarrollador pueda crear un tema defectuoso en el futuro y facilitando el autocompletado de código cuando éstos se utilicen, dado que tienen una estructura conocida antes de su compilación.

Continuando con la carpeta «src», se encuentran tres archivos muy importantes:

- index.html: En él se carga la fuente personalizada del proyecto, Montserrat, se incluye una animación mientras la aplicación se carga y se especifica el punto de partida o raíz de toda la aplicación Angular, señalizada con la etiqueta «<app-root></app-root>».
- main.ts: Carga el módulo raíz de la aplicación «app».

- styles.scss: Contiene estilos generales y aplica la fuente de letra personalizada para toda la aplicación.

8.2.2 Módulo Core

En el apartado de diseño se ha establecido que este módulo contenga exclusivamente servicios de ámbito general y accesibles desde cualquier lugar de la aplicación.

Utilizando el comando «ng generate service *nombre_del_servicio*» en el directorio «/core» de la aplicación, se crea un nuevo servicio a partir de una plantilla. Este nuevo servicio creado, es una clase muy similar a una clase pura de TypeScript, cuya principal diferencia es que se instancia una única vez durante todo el transcurso de uso de la aplicación. Dicha instancia se inyecta en aquellos componentes, servicios o guards que la utilicen. Por tanto, los servicios permitirán establecer una única fuente de verdad (*single source of truth*), sincronizando la información entre los componentes que la utilicen.

Siguiendo el diseño propuesto, se ha establecido una estructura de servicios y clases enumerada y explicada a continuación.

NOMBRE	auth
TIPO	Servicio
PROPÓSITO	Envío de peticiones al servidor para comprobar la existencia y validez del ticket recibido en la ruta de la página. Coordinar el funcionamiento del guard de seguridad, indicándole si el usuario tiene permiso para proseguir con el uso de la aplicación. Si no se puede comprobar el ticket o el usuario no es válido, redirigir a otras pantallas de alerta. Comprobar que el usuario puede realizar la operación a la que ha accedido con su enlace.
MÉTODOS	setTicket(ticket: string) → Establece y almacena el ticket proporcionado. getTicket( ): string → Devuelve el ticket almacenado. findTicket(ticket: string): Observable<boolean> → Envía al servidor la petición de búsqueda del ticket. Devuelve un booleano indicando *true* si el ticket existe y es válido y *false* en caso contrario. setGuardLoading(isLoading: boolean) → Establece el estado del guard, indicando si éste esta comprobando la validez del ticket, e índicándoselo al usuario con una animación o si ya ha acabado sus comprobaciones. getGuardLoading( ): Observable<boolean> → Obtiene el estado del guard. setRequestType(string [ ]) → Establece el tipo de acción permitida del ticket (introducción, cambio o recuperación de contraseña). getRequestType( ): Observable<string[ ]> → Obtiene el tipo de acción permitida del ticket.

Tabla 11: Servicio auth

En la mayoría de servicios creados y utilizados en esta apliación se ha hecho uso de un paradigma de programación conocido como programación reactiva. La programación reactiva trata de controlar los flujos de datos y su propagación a través de la aplicación.

En este paradigma, la interfaz va adaptándose a la información que dispone y reaccionando de forma asíncrona a los datos que recibe según van apareciendo éstos. Las peticiones a los servidores se consideran eventos asíncronos a los que los componentes o servicios se subscriben y, una vez obtenida la respuesta de la petición, se ejecuta el fragmento de código establecido en cada uno de los componentes o servicios que se han suscrito a ese flujo de datos.

En la Figura 15 se ilustra el funcionamiento de este paradigma con la función *findOpenTicket*. Pueden existir muchos componentes y servicios subscritos al valor de una variable, pero solo es necesario que un componente o servicio realice la petición que produzca un cambio en esa variable, que se propagará por igual a todos los componentes o servicios subscritos. Cada uno de estos componentes o servicios establecen una porción de código que se ejecutará en caso de recibir nuevos datos y los transformará en otros a los que, a su vez, pueden ser suscritos terceros componentes, creando flujos o cadenas de información que se propagan por toda la aplicación.

Para aplicar este paradigma a esta aplicación Angular, se utiliza la librería RxJS (*Reactive Extensions Library for JavaScript*). Existen dos objetos utilizados de gran relevancia:

- Observable: Es un objeto proporcionado a los componentes o servicios que desean reaccionar a un flujo de información. Con este objeto, dichos componentes o servicios se subscriben a los cambios en el flujo de datos. Al subscribirse se proprociona al observable una función que se ejecutará con cada nuevo cambio en el flujo de datos.
- BehaviorSubject: Este objeto se utiliza en los servicios que desean establecer un flujo de datos que sea utilizado en otros lugares de la aplicación. Puede establecerse o no un valor inicial (que puede ser booleano, numérico o cualquier otro tipo). Cada vez que se desee propagar un cambio en el último valor almacenado en el objeto se utiliza la función next(valor nuevo). Este objeto se entrega a cada uno de los componentes o servicios en forma de Observable (que no permite emitir nuevos datos, por seguridad). En el momento en el que se establece un valor nuevo, quien esté subscrito lo recibirá y actuará en consecuencia.

En el caso de la Figura 15, se almacena la validez del ticket en un objeto de tipo *BehaviorSubject*. Cuando los componentes llaman a la función *findOpenTicket* reciben un *Observable* de esa variable y se subscriben a él. Cuando se realiza la petición, el resultado de ésta, cuando llegue, será recibido por todos ellos por igual, reflejándolo en la interfaz o ejecutando otras funciones a partir de la validez del *ticket*.

La librería RxJS también permite la realización de transformaciones y el filtrado de la información por parte de los componentes subscritos a los *observables*.

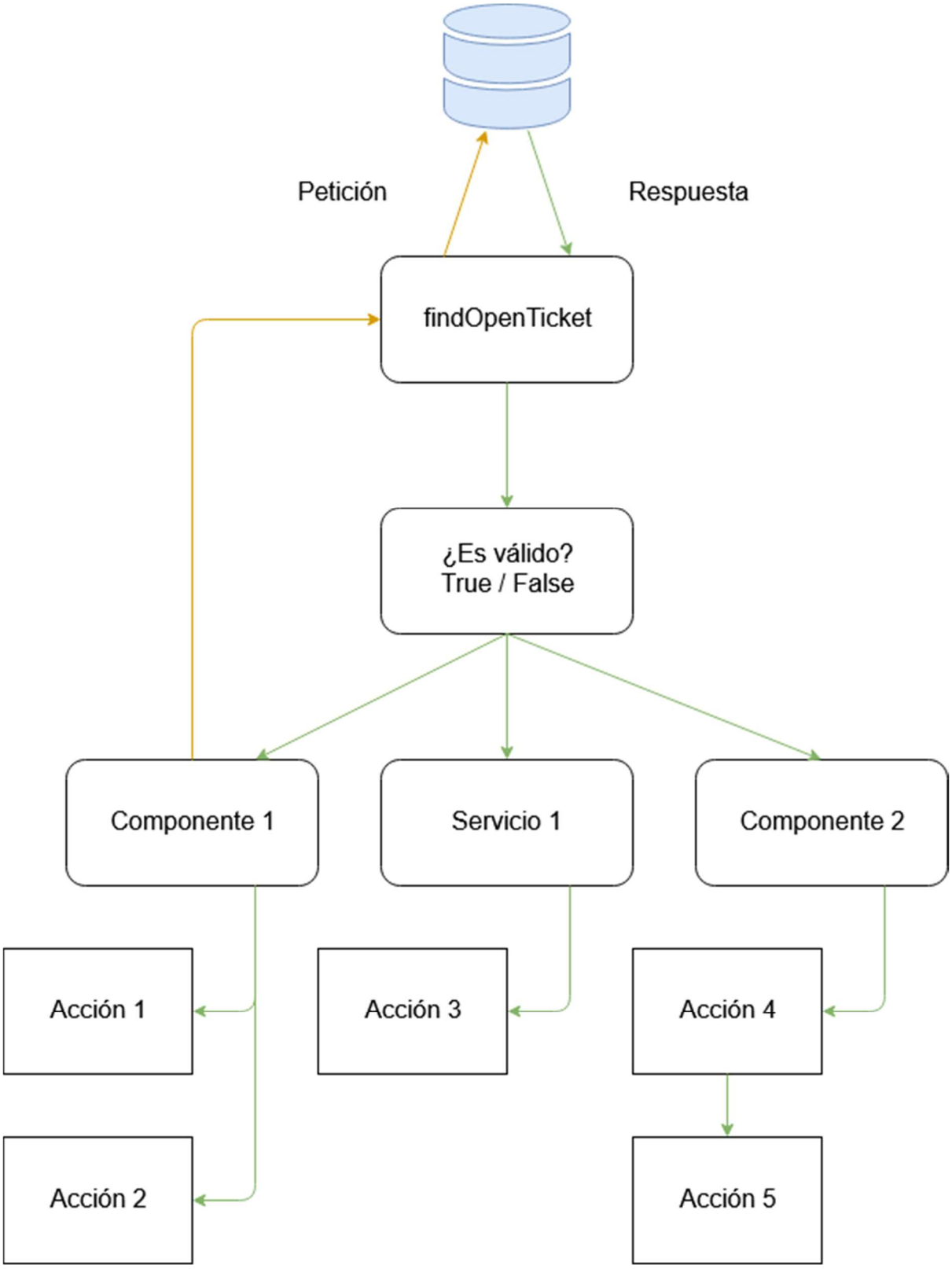

Figura 15: Diagrama de un evento asíncrono en programación reactiva

NOMBRE	auth
TIPO	Guard
PROPÓSITO	Comprueba la validez del ticket y permite o no al usuario proseguir con el uso de la aplicación. En caso de no ser válido el ticket, redirige al usuario a una pantalla de alerta.
MÉTODOS	canActivate( ): boolean → Este método es obligatorio en un guard y establece con un *true* si el usuario puede proseguir o un *false* si el usuario no puede proseguir.

Tabla 12: Guard auth

El guard de la Tabla 12 se ha creado con el comando «ng generate guard *nombre_del_guard*».

NOMBRE	HMAC
TIPO	Clase
PROPÓSITO	Firmar peticiones para asegurar que el contenido no ha sido modificado durante la comunicación con HTTPS.

Tabla 13: Clase HMAC

NOMBRE	Http-Interceptor
TIPO	Clase
PROPÓSITO	Interceptar todas las comunicaciones con el servidor y utilizar la librería HMAC para añadir una cabecera con el contenido firmado.

Tabla 14: Clase Http-Interceptor

NOMBRE	theme
TIPO	Servicio
PROPÓSITO	Carga de imagen, logo, colores corporativos nombre e icono de página y su puesta a disposición de otros componentes.
MÉTODOS	setBrowserTitle(sectionName: string) → Establece el título de la página. upperCaseFirstChars(sentence: string): string → Cambia la primera letra de cada palabra a mayúscula. setFavicon(image: string) → Establece el icono proporcionado como icono de la página. getTransitionDuration( ): number → Obtiene la duración en milisegundos de las transiciones de la aplicación, de modo que todas estén coordinadas. getActiveTheme( ): Theme → Obtiene el tema establecido. loadActiveTheme( ) → Carga los colores del tema en variables CSS para ser utilizados en otras partes de la aplicación sin tener que importarlas.

Tabla 15: Servicio theme

NOMBRE	translations
TIPO	Servicio
PROPÓSITO	Detección del idioma del usuario si éste no puede conocerse con el ticket. Carga de traducciones del servidor. Cambio del lenguaje de los textos mostrados al usuario.
MÉTODOS	overrideTranslationsLanguageCode(languageCode: string) → Cambia el lenguaje de los textos. getTranslationsLanguageCode( ): string → Obtiene el lenguaje establecido. setPreferredTranslationsLanguage( ) → Compara los lenguajes del navegador con los de la lista de lenguajes disponibles y establece como lenguaje principal la primera coincidencia. getBrowserLanguageCodes( ) → Obtiene la lista de lenguajes disponibles en el navegador del usuario, ordenados por prioridad. fetchAvailableLanguages( ) → Realiza una petición al servidor para obtener la lista traducida de lenguajes disponibles. getAvailableLanguages( ) → Obtiene la lista de lenguajes.

Tabla 16: Servicio translations

El módulo Core ha sido incluído en este capítulo dado su carácter general. Sin embargo, los módulos Static, Features y Shared serán pormenorizados en los siguientes apartados junto con los prototipos de la aplicación.

8.3 Esqueleto de la aplicación

El esqueleto de la aplicación se localiza en un solo componente dentro del módulo Static. Este componente, llamado «main.component.ts» tiene como función principal la carga del tema activo con las rutas del logo y la imagen corporativos. En la puede observarse el prototipo del esqueleto de la aplicación para la marca Volkswagen.

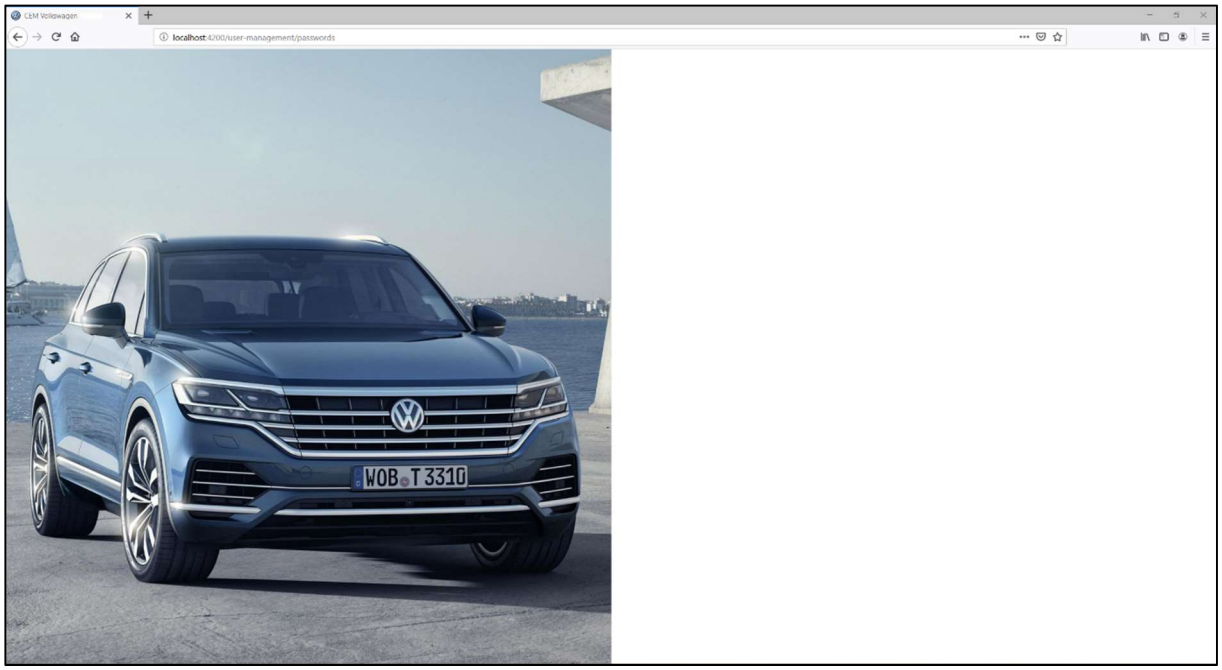

Figura 16: Prototipo del esqueleto de la aplicación web (versión de escritorio)

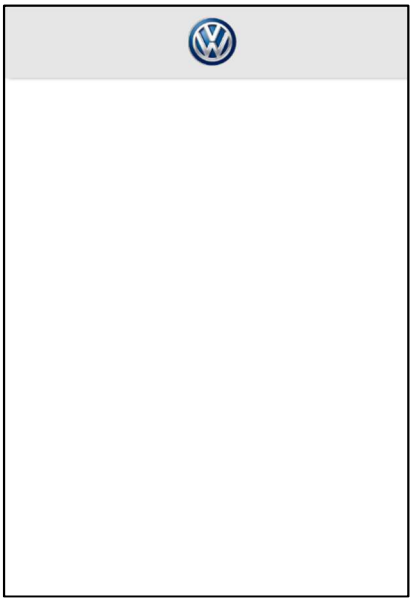

Figura 17: Prototipo del esqueleto de la aplicación web (versión móvil)

8.4 Páginas

Las páginas descritas en los siguientes apartados están formadas por un componente y un servicio cada una. Usando los componentes reutilizables creados en el módulo Shared, crean cada uno de los procesos o experiencias disponibles para el usuario.

Los componentes del módulo Shared no almacenan ningún tipo de información y la reciben toda a partir de lo que se conoce en Angular como «Input» y en caso necesario, devuelven valores con un «Output». Un «Input» y un «Output» son anotaciones utilizadas junto con una variable que la marcan como una entrada de información o una salida de información de un componente. Puede verse un ejemplo de un «Input» y un «Output» en la Figura 18.

```
@Input() disabled: boolean;
@Output() private onPasswordButtonClick: EventEmitter<any>;
```

Figura 18: Ejemplo de un Input y un Output

Nótese que a diferencia de un «Input», un «Output» devuelve los valores deseados emitiendo eventos que invocan otra función en el componente que utiliza a éste. Su comportamiento es similar al de un observable.

Para utilizar los componentes creados en el módulo Shared (o en cualquier otro), es necesario importar el módulo de lo contiene (en este caso Shared) al módulo que lo va a utilizar. Después, en un archivo *.html* se escribirá una etiqueta con el siguiente formato siguiendo el ejemplo de la figura anterior:

```
<nombreDelComponente [disabled]=”false” (onPasswordButtonClick)=”submit( )”>
Otros componentes (si los hubiera)
</nombreDelComponente>
```

A la hora de crear cada componente se ha decidido evitar, en la medida de lo posible, crear componentes con funciones similares y con diferencias sutiles. Se ha intentado en todo momento extender los componentes creados para que se comporten de una u otra forma dependiendo de la información recibida.

A continuación en la Tabla 17, se muestra una lista con todos los componentes creados en el módulo Shared. En la primera columna se encuentra un índice de componente que será visible junto al mismo en las imágenes de los prototipos que aparecerán en los siguientes apartados.

ÍNDICE	NOMBRE	FUNCIONALIDAD
1	button	Botón genérico con capacidad de ser bloqueado.
2	choser	Grupo de botones en el que solo uno de ellos puede estar seleccionado.

3	error-message	Mensaje en color rojo con un icono indicando error.
4	form	Formulario que acoge otros elementos de tipo input.
5	info	Contiene un título principal grande, un subtítulo y un botón de redirección. Permite informar al usuario de una situación o alertarle.
6	input	Entrada de texto plano o de contraseñas, con capacidad de mostrarlas u ocultarlas en el último caso. Puede ser bloqueado.
7	label	Subtítulo de tamaño normal con posibilidad de cambiar su color.
8	label-light	Subtítulo con una fuente de letra más delgada y de menor tamaño, ademite cambiar su color.
9	language	Botón con el icono de la bandera de un país, dispone de un bocadillo indicando el nombre del país cuando el usuario posa el puntero encima del botón.
10	language-button	Botón con el icono de la bandera de un país.
11	languages-container	Contenedor para agrupar un conjunto de elementos del tipo language.
12	logo	Logotipo de la empresa. Se muestra sólo en la versión móvil de la aplicación.
13	rule	Regla de creación de contraseña. Contiene un icono, que puede indicar que la regla se cumple o un icono que indique que la regla se comprueba en el servidor.
14	rules-container	Contenedor para agrupar un conjunto de elementos de tipo rule.
15	spinner	Animación de carga. Indica al usuario que debe esperar a que finalice algún tipo de proceso.
16	title	Texto de tamaño grande, admite cambiar su color.
17	transition-container	Contenedor de tipo general que puede ocultarse y aparecer con animaciones de aparición y desvanecimiento.

Tabla 17: Componentes del módulo Shared

Todos los componentes del módulo Features siguen un patrón muy similar. Establecen las propiedades que van a utilizar en una interfaz con los tipos de cada una fijados, añadiendo un paso extra de seguridad. Entonces, se crea una variable inscrita a esa interfaz llamada «InitialState», donde se dan valores iniciales a esas variables.

En el caso de los componentes de cambio, introducción y recuperación de contraseña, al ejecutarse, se guarda el valor de «InitialState» en una variable llamada «State», a la que accederán

todas las funciones y componentes utilizados cuando necesiten conocer el valor de alguna propiedad del «State». Para cambiar el valor de alguna de las propiedades del «State», se ha creado una utilidad que muta el objeto «State» con el valor nuevo, en vez de copiarlo, que consume más recursos del ordenador del usuario.

Continuando con la ejecución del componente, se cargan las traducciones, se cambia el título de la página, se obtiene el tema de la aplicación, se autentifica el *ticket* (en los casos en los que haya), se obtienen las reglas de creación de contraseñas (en los casos necesarios) y se obtiene la lista traducida de lenguajes disponibles en la plataforma.

Por último, se declaran los campos de los formularios.

En el caso del componente que crea las páginas de alerta, simplemente se carga el tema de la aplicación y se comprueba el argumento recibido del archivo de rutas, mostrando unos textos u otros.

8.4.2 Cambiar contraseña

En las siguientes figuras puede observarse el prototipo de la página de cambio de contraseña. Los textos aparecen en inglés debido a que, en el momento de redacción de este documento, no se ha podido obtener a tiempo las traducciones encargadas a una empresa de traducción. Sin embargo, cuando se reciban, serán implementadas y podrá cambiarse el lenguaje haciendo uso del menú de cambio de lenguaje.

Un pequeño cambio, aparte de algunos cambios a nivel estético, es la creación de un botón en cada campo del formulario que permite mostrar u ocultar la contraseña que se ha escrito (icono de un ojo).

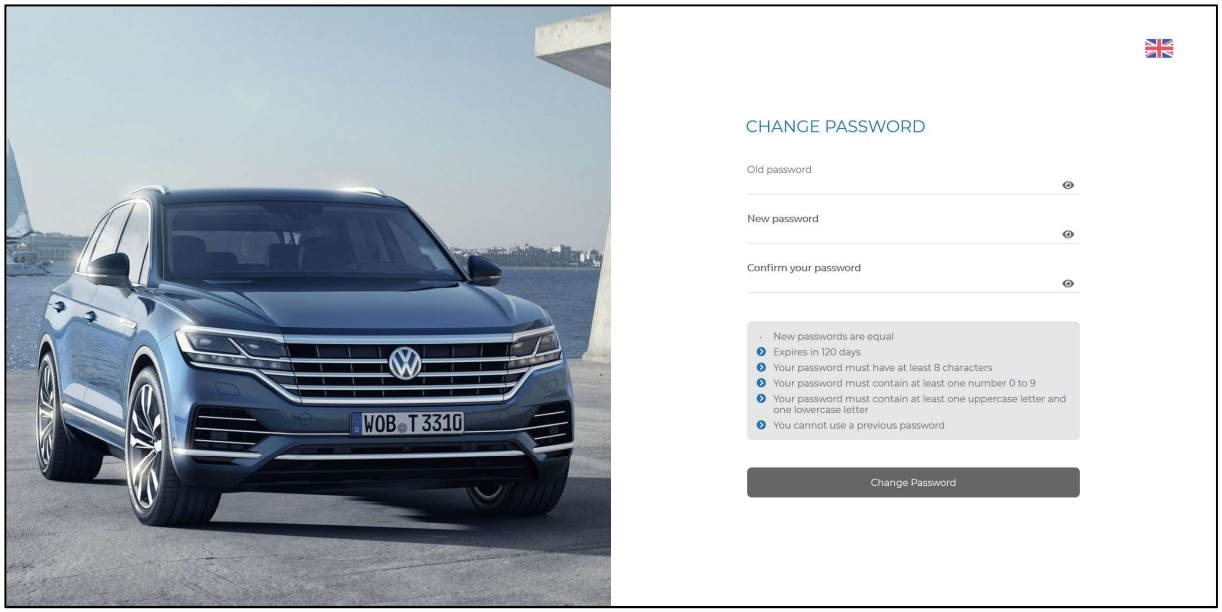

Figura 19: Prototipo de la página de cambio de contraseña de la aplicación web

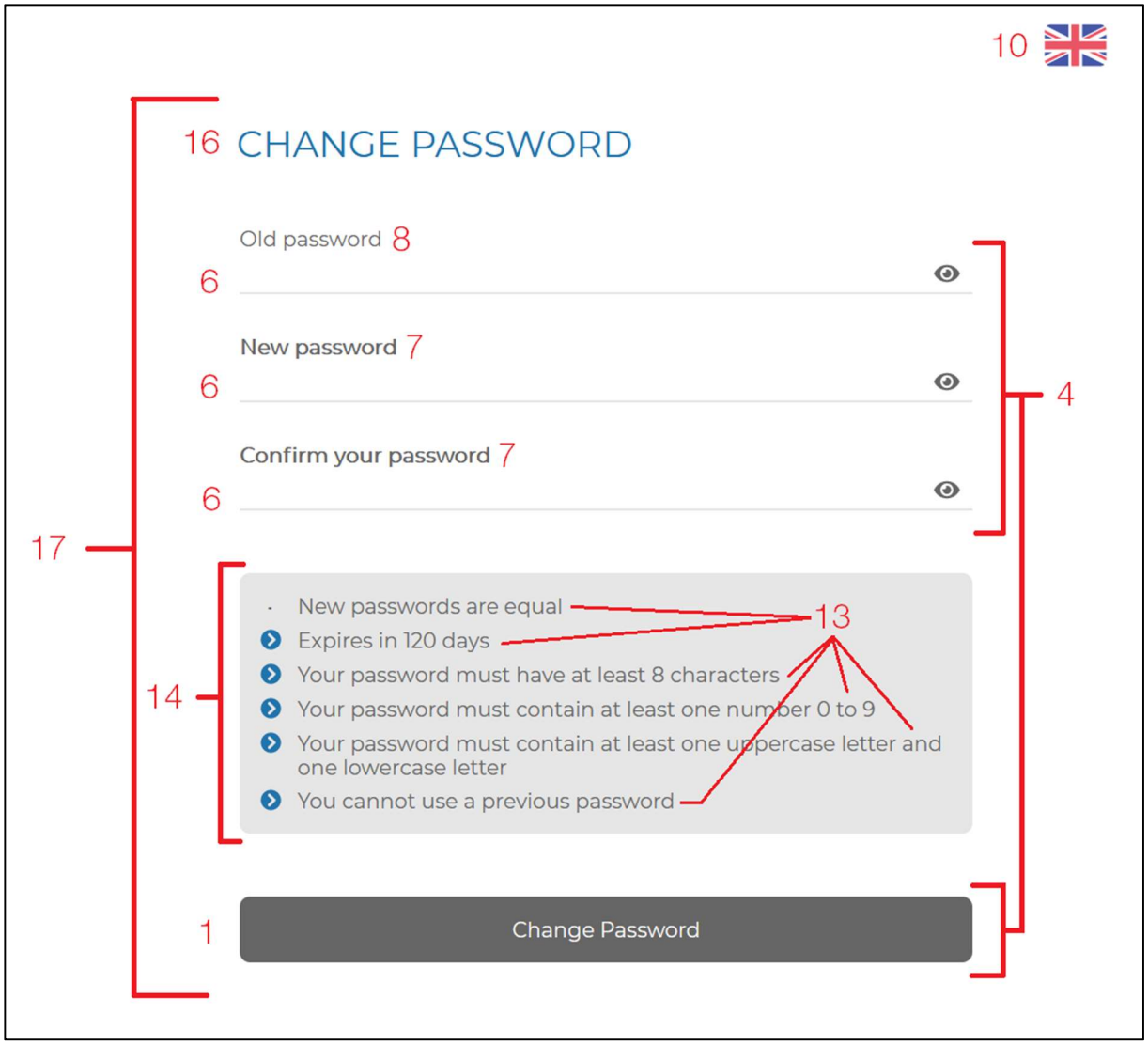

Figura 20: Ampliación del área de cambio de contraseña

En la Figura 21 se ha dispuesto un prototipo del área de cambio de lenguaje. Como se ha comentado anteriormente, en el momento de redacción de este documento no se ha podido implementar completamente la funcionalidad de cambio de lenguaje. Por tanto, lo mostrado en esta figura corresponde a una simulación del aspecto final de esta pantalla.

En la Figura 17 puede observarse el componente 12, logo, el logotipo de la empresa, en este caso, Volkswagen.

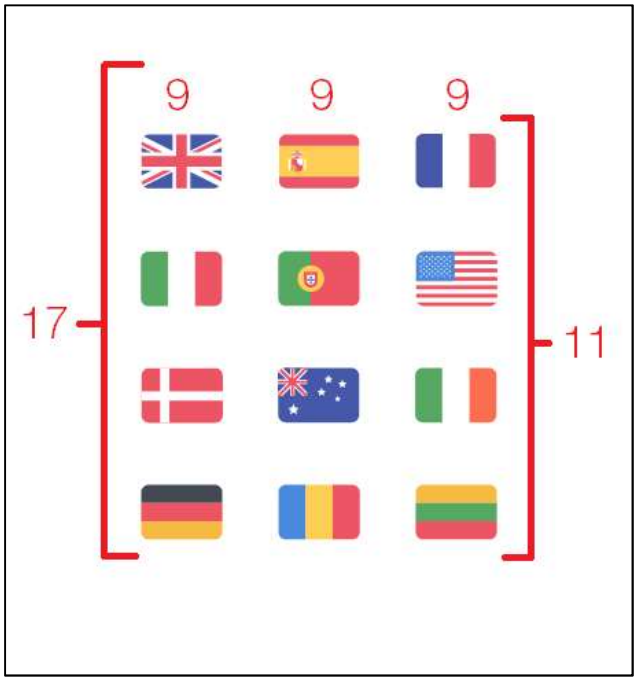

Figura 21: Ampliación del área de cambio de lenguaje

8.4.3 Nueva contraseña

Esta página difiere de su diseño original en el mismo grado que la página del apartado anterior. Cabe recordar que, aunque solo se muestren una vez en otros apartados, tanto la página de cambio de lenguaje como la página de éxito están disponibles en las páginas de introducción, cambio y recuperación de contraseña.

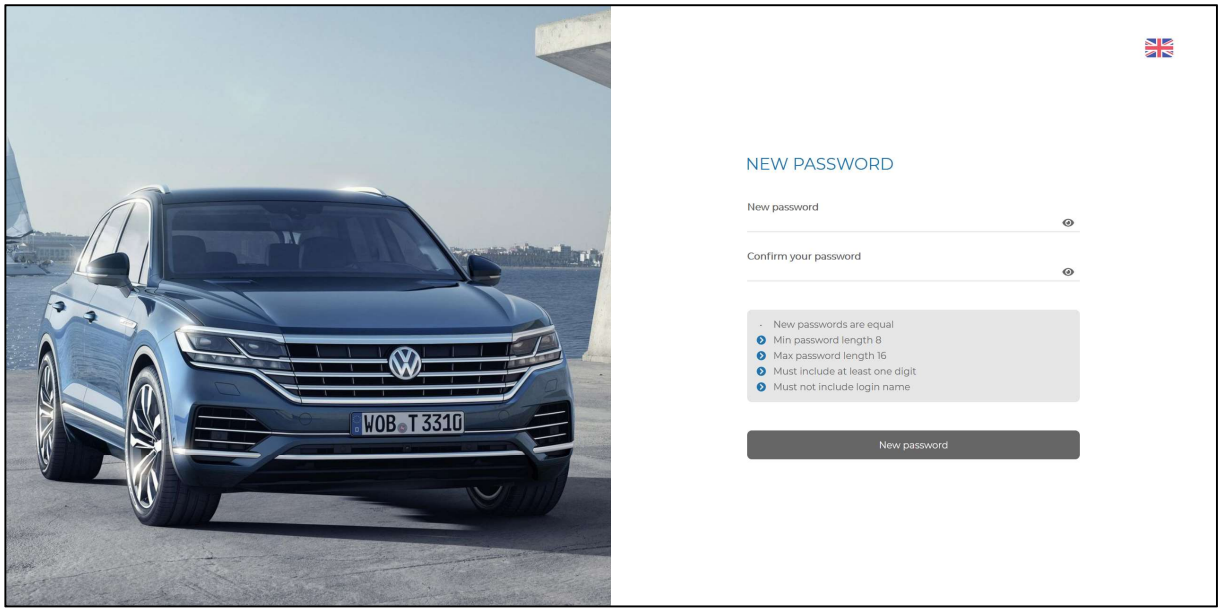

Figura 22: Prototipo de página de introducción de contraseña

8.4.4 Recuperar contraseña

En la Figura 23 se muestra el prototipo de la página de recuperación de contraseña. En este prototipo se ha cambiado la imagen y los colores corporativos para que el lector pueda percibir los cambios estéticos que se producen al cambiar de marca automovilística.

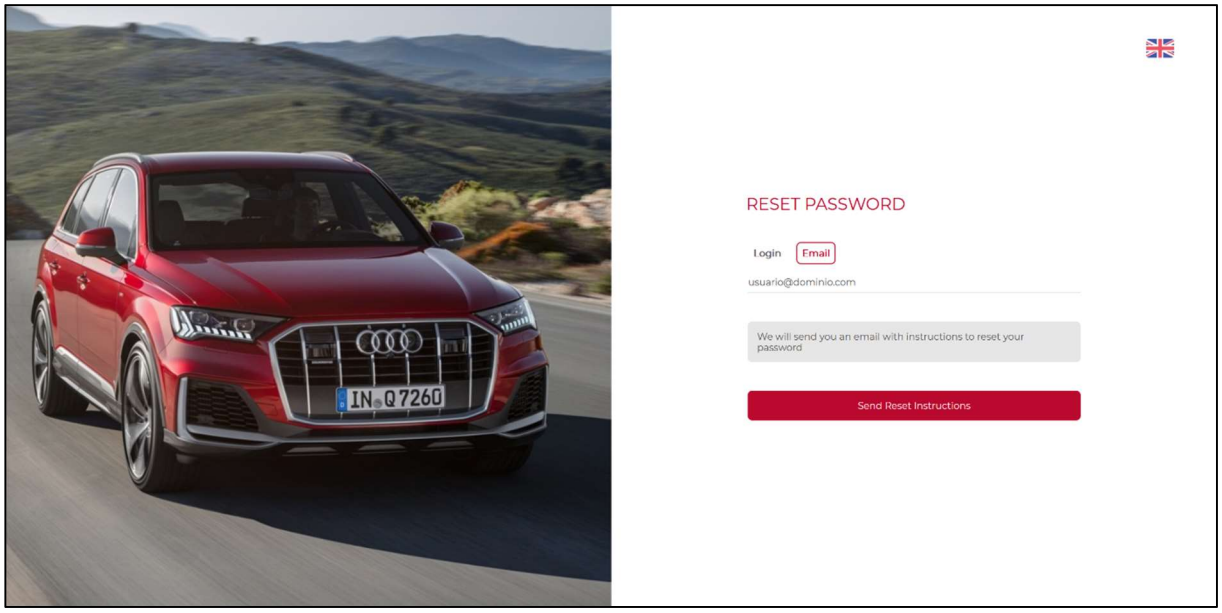

Figura 23: Prototipo de página de recuperación de contraseña

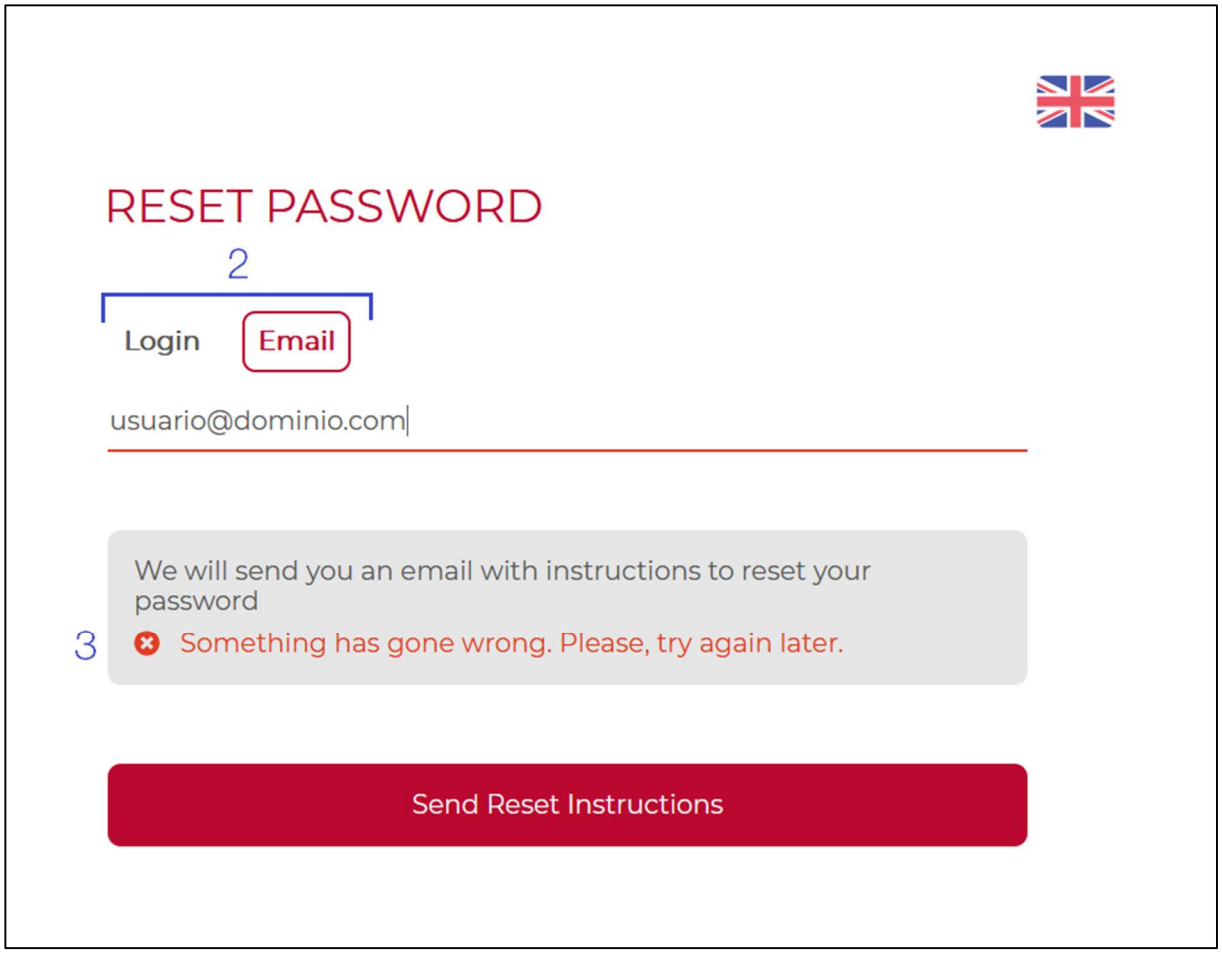

Figura 24: Ampliación de la página de recuperación de contraseña (Error)

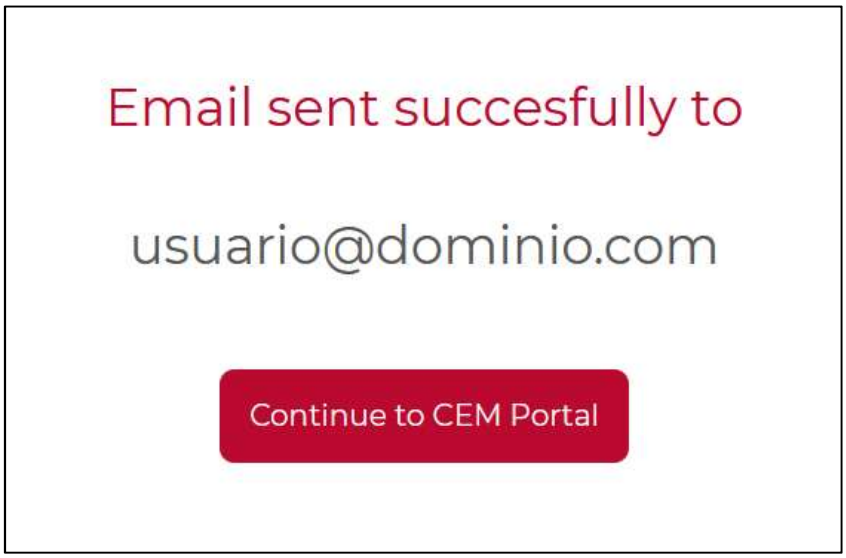

Figura 25: Ampliación de la pantalla de éxito de la página de recuperación de contraseña

8.4.5 Páginas de alerta

Para las páginas de alerta se ha ejecutado el diseño marcado anteriormente. Para evitar que la página pudiese quedar demasiado vacía, se ha optado por añadir un fondo con un patrón en gris, reutilizable en todas las marcas debido a su carácter abstracto y aséptico. A continuación se muestran los prototipos de los distintos casos de uso.

Figura 26: Ampliación del prototipo de página de alerta (Error)

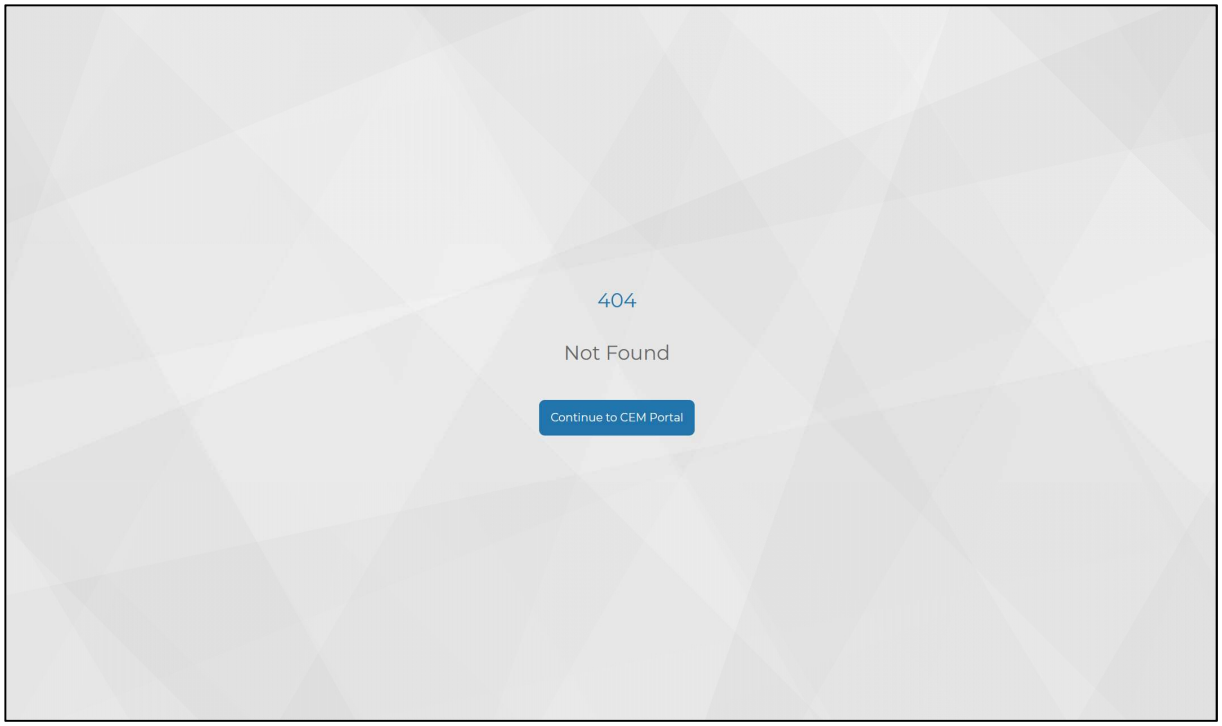

Figura 27: Prototipo de página de alerta (Página no encontrada)

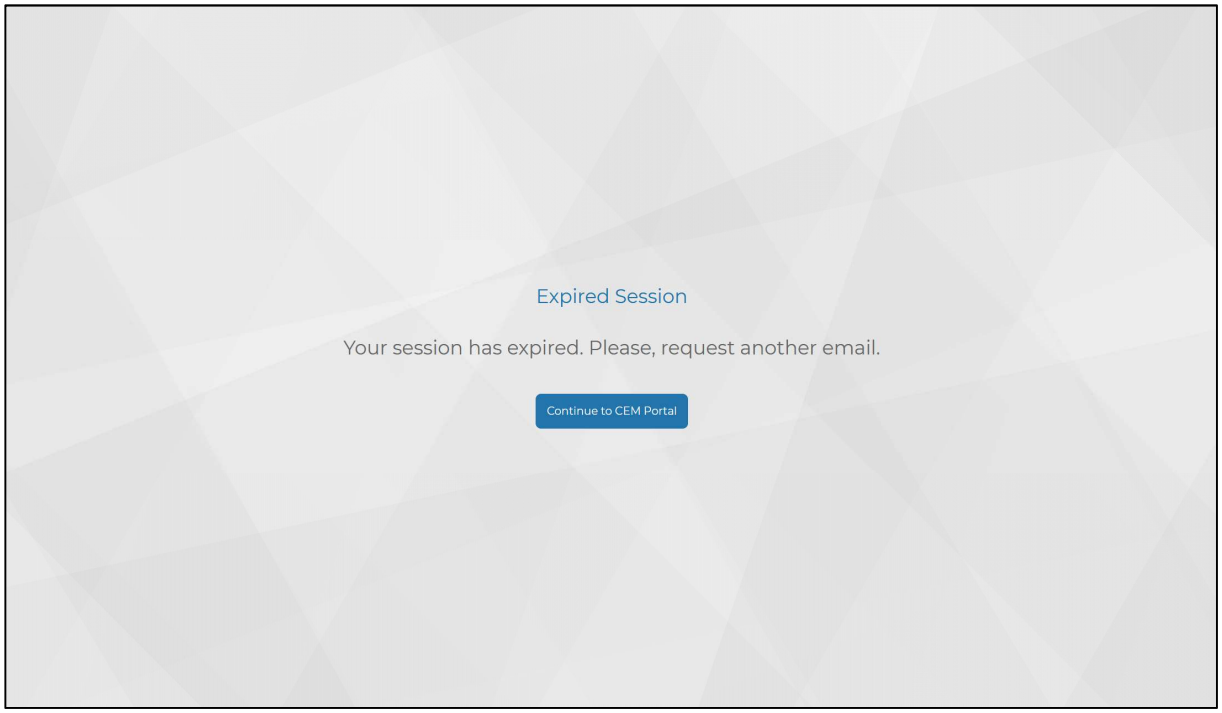

Figura 28: Prototipo de página de alerta (Sesión caducada)

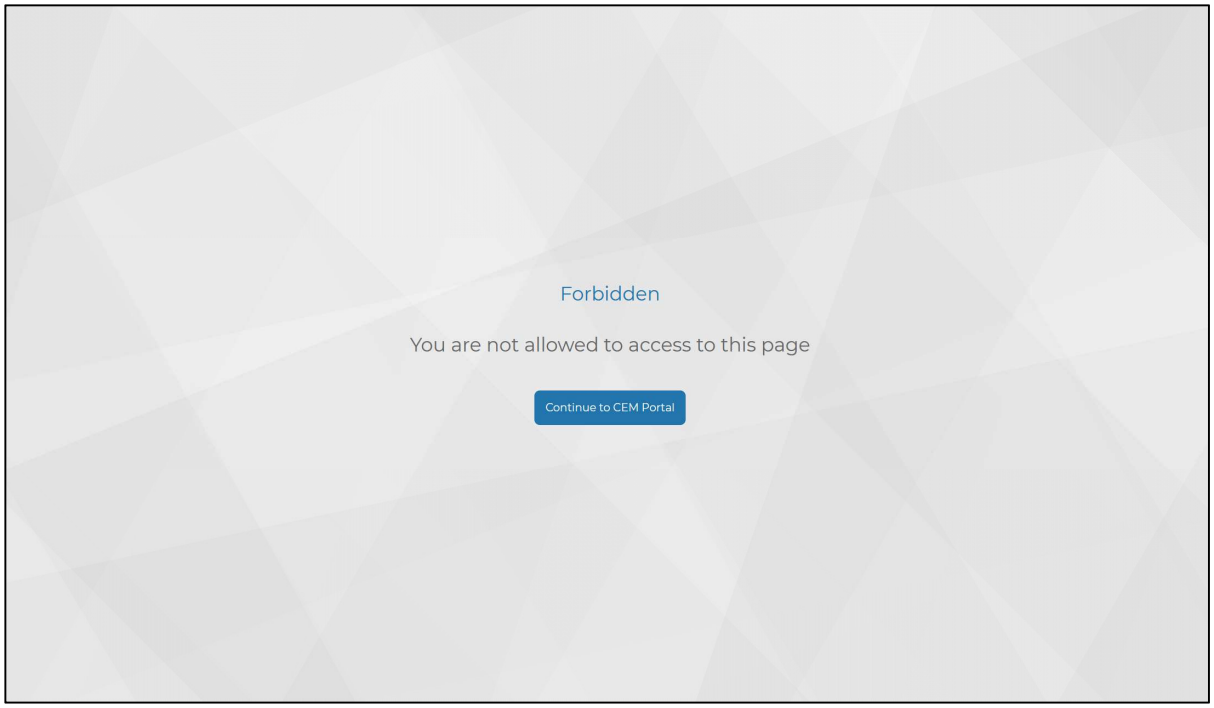

Figura 29: Prototipo de página de alerta (Acceso no permitido)

9
Desarrollo de las adaptaciones al servidor

A la hora de realizar las adaptaciones al servidor, el primer paso es descargar el código que va a ser modificado. En la empresa, ODEC, se hace uso de la herramienta Subversion para la gestión de cambios y versiones de los proyectos. El funcionamiento de Subversion es similar al de Git y, para acceder a los proyectos, se necesita descargar en Eclipse el complemento Subclipse para poder acceder a la perspectiva *SVN Repository Exploring*.

Dentro de esa perspectiva, en el panel izquierdo, haciendo clic con el botón secundario puede añadirse una nueva localización de repositorio (pegando el enlace del mismo). En el caso concreto de este proyecto, se escoge la rama de desarrollo de las *APIs user* y *messages*.

Una vez descargados los dos proyectos, accediendo a la perspectiva JavaEE, en el panel izquierdo, haciendo clic con el botón secundario en cada uno de ellos, se importan y se convierten a un proyecto Maven, una herramienta que facilita el uso y creación de programas Java a nivel empresarial.

A continuación, haciendo clic con el botón secundario del ratón en los dos proyectos, bajo el menú Maven se actualiza por separado cada uno de los proyectos, recopilando y actualizando todas sus dependencias. Entonces, haciendo clic con el botón secundario sobre el archivo *pom.xml*, de cada uno de los dos proyectos, bajo el menú Maven, se ejecuta la acción *maven install*, que los compila y los guarda en el ordenador del desarrollador.

Realizados todos estos pasos, se procede a comenzar el desarrollo de las adaptaciones a ambas *APIs*.

A nivel general, en las dos *APIs* se aplica una arquitectura por capas, como se ha comentado en el apartado de diseño. El propósito de este diseño en capas es la separación de responsabilidades, de arquitecturas y de patrones de diseño, de modo que si, por ejemplo, el patrón REST dejase de utilizarse en el futuro, no sería necesario realizar modificaciones a otras partes de la infraestructura.

Cuando el servidor recibe una petición, ésta va dirigida a alguno de los métodos publicados por el servidor. Dichos métodos se publican en la capa REST, aceptando una serie de argumentos, en la ruta de la petición o en el cuerpo de la misma, que condicionan el contenido de la respuesta o las acciones que se llevarán a cabo.

Con los argumentos recibidos, el método de la capa REST hará uso de aquellos que estime oportunos (normalmente todos) y llamará a una función de la capa EJB.

La función de la capa EJB comienza entonces el proceso concreto para el que ha sido diseñado y se presentan dos posibles situaciones:

- El proceso finaliza correctamente. La capa REST recibe el resultado del método de la capa EJB y genera y envía un objeto de la clase Response que contiene el resultado del método EJB (si hubiera algo que recibir) y le asigna el código 200, OK.
- El proceso encuentra algún error que le impide finalizar. La capa EJB o las funciones que ésta utilice lanzan una excepción. En ese momento, se para la ejecución del método de la capa EJB y se vuelve a la capa REST, donde se detecta el tipo de excepción y se genera un objeto de la clase Response con el código de error adecuado. Si no se pudiera detectar el propósito u origen de la excepción, se generería una respuesta con el código de error 505, Internal Server Error.

9.1 Capa REST

Para publicar un método en la capa REST es necesario incluir una serie de anotaciones que indiquen las características del método y ayuden a documentarlo.

La primera anotación indica el tipo de peticiones que acepta el método, como GET, POST, PUT o DELETE. Un ejemplo sería la siguiente anotación: `@GET`.

Para indicar la ruta necesaria para acceder al método publicado es necesaria la anotación `@Path("/getRulesPasswordFromListWithTicket")`.

En este proyecto se utiliza la herramienta Swagger, que genera automáticamente páginas con documentación y ofrece la posibilidad de, a través de una sencilla interfaz, realizar peticiones al servidor manualmente.

La información relacionada con el propósito o funcionalidad del método se plasma en la anotación `@ApiOperation(value = "Obtener las reglas para introducir un nuevo password a partir de un ticket, admite sobreescribir el lenguaje y la variante")`.

A continuación de la anotación anterior, se introducen en un grupo todas aquellas respuestas posibles del método, junto a su significado y a la información que puedan devolver.

```
@ApiResponses({

	@ApiResponse(code=HttpServletResponse.SC_OK,  response=String.class,
	responseContainer="List", message="Devuelve la lista de reglas aplicables al
	usuario"),

	@ApiResponse(code=HttpServletResponse.SC_UNAUTHORIZED, message="El ticket
	está cerrado"),

	@ApiResponse(code=HttpServletResponse.SC_NOT_FOUND, message="No se ha
	encontrado el ticket"),

	@ApiResponse(code=HttpServletResponse.SC_INTERNAL_SERVER_ERROR,
	message="Excepción no controlada por el servidor"),

})
```

Una vez añadidas estas anotaciones, se introduce el nombre del método, así como el tipo de la respuesta, y los argumentos que éste va a recibir.

Utilizando la anotación `@ApiParam(name = "ticket", required = true, value = "Ticket")` encima de los argumentos declarados se indica a Swagger la existencia de este argumento, si es obligatorio y el significado del argumento.

Los argumentos recibidos en la ruta de la petición se llaman *QueryParams* y se indican utilizando una anotación como la siguiente: `@QueryParam("ticket")Ticket ticket`. Los argumentos recibidos a través del cuerpo de la petición se conocen como `@FormParam("ticket")Ticket ticket`. El *ticket* recibido desde la aplicación *web* se almacena en la clase Ticket que consta exclusivamente de una variable de tipo String y un método que permite obtener el valor del *ticket*. De esta manera, se protege el contenido del ticket de modificaciones accidentales y maniefiesta la estructura de los métodos de forma más sencilla para otros desarrolladores.

En los métodos de la capa REST se utilizan las expresiones *try* y *catch* para la ejecución de los métodos de la capa EJB. Dentro del área formada por los corchetes de la expresión *try* se introduce el código que debe ejecutarse si el proceso de realiza sin errores (en este caso, la respuesta con el código 200, OK) y si ocurre algún error se ejecuta el código localizado entre los corchetes de la expresión *catch*. Dentro de esta última expresión, se identifica el tipo de excepción y se genera la respuesta correspondiente. Puede verse un ejemplo completo de todo lo comentado en estos últimos párrafos a continuación, en la Figura 30.

```
@GET
@Path("/getRulesPasswordFromListWithTicket")
@ApiOperation(value = "Obtener las reglas para introducir un nuevo password a partir de un ticket, admite sobreescribir el lenguaje y la variante")
@ApiResponses({
    @ApiResponse(code=HttpServletResponse.SC_OK,  response=String.class, responseContainer="List", message="Devuelve la lista de reglas aplicables al usuario"),
    @ApiResponse(code=HttpServletResponse.SC_UNAUTHORIZED, message="El ticket está cerrado"),
    @ApiResponse(code=HttpServletResponse.SC_NOT_FOUND, message="No se ha encontrado el ticket"),
    @ApiResponse(code=HttpServletResponse.SC_INTERNAL_SERVER_ERROR, message="Excepción no controlada por el servidor"),
})
public  Response getRulesPasswordFromListWithTicket(
        @ApiParam(name = "ticket", required = true, value = "Ticket")
        @Valid
        @QueryParam("ticket")Ticket ticket,
        @ApiParam(name = "language", required = false, value = "Código del lenguaje")
        @QueryParam("language")String language,
        @ApiParam(name = "variant", required = false, value = "Código de la variante del lenguaje")
        @QueryParam("variant")String variant
        )
{

    try
    {
        return Response.status(HttpServletResponse.SC_OK).entity(getUserEJB().getRulesPasswordToUserList(ticket, language, variant)).build();
    }
    catch (EJBException ex)
    {
        LOG.error(ex);
        return Response.status(Status.INTERNAL_SERVER_ERROR).entity(JSONResponse.message("internal server error")).build();
    }
    catch (Exception ex)
    {
        if (ex instanceof ClosedUserRequestException) {
            return Response.status(Status.UNAUTHORIZED).entity(JSONResponse.message("unauthorized")).build();
        }

        if (ex instanceof UnknownUserRequestException)
            return Response.status(Status.NOT_FOUND).entity(JSONResponse.message("not found")).build();

        LOG.error(ex);
        return Response.status(Status.INTERNAL_SERVER_ERROR).entity(JSONResponse.message("internal server error")).build();
    }
}
```

Figura 30: Ejemplo de método de la capa REST

Las pruebas de funcionamiento de cada uno de los métodos creados según la propuesta de diseño se han realizado la herrameinta citada, Swagger. Esta herramienta se utiliza con un navegador *web*, accediendo a una página que se genera cuando el servidor está arrancado. Dentro de la página se puede acceder a todos los métodos de la *API* e interactuar con ellos.

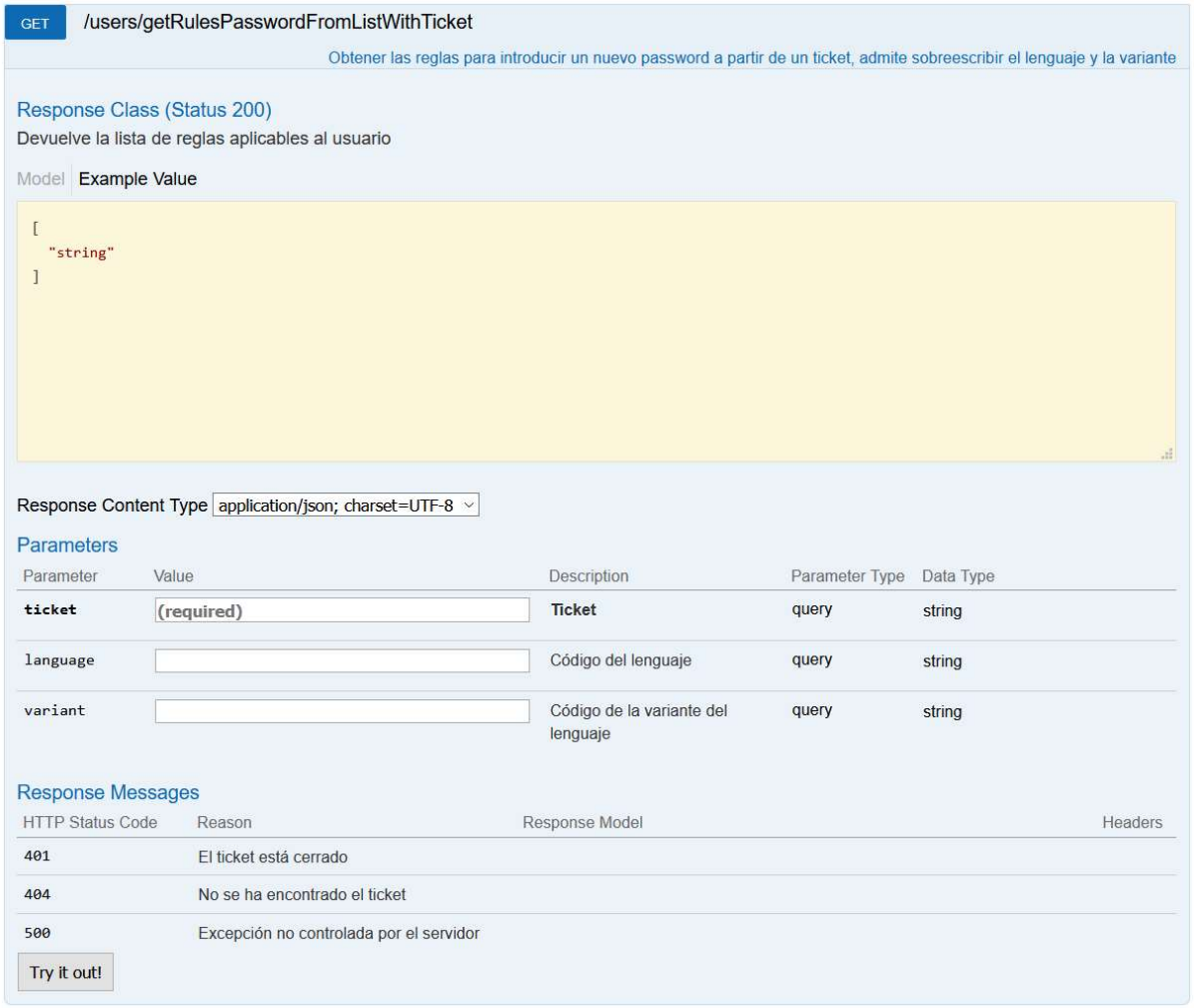

Figura 31: Captura de pantalla de la herramienta Swagger

9.2 Capa EJB

Los métodos de la capa EJB realizan principalmente la extracción de la información requerida de las bases de datos de la empresa.

Cuando se produce algún tipo de error, bien sea por la lógica misma del método o bien por una excepción lanzada por alguno de las funciones utilizadas en el mismo, ésta se emite a la capa REST, como ya se ha comentado.

Generalmente, en los métodos que lo requiere, se utiliza el *ticket* para obtener toda la información necesaria del usuario, evitando que la aplicación web deba almacenarla y proporcionando una mayor seguridad al sistema, pues hay menos datos sensibles siendo comunicados a través de Internet.

Obtenidos los datos personales del usuario se pueden realizar con total seguridad las operaciones necesarias, normalmente en forma de sentencias SQL almacenadas en otras partes del programa y creadas por otros desarrolladores de la empresa.

Los métodos de la capa EJB están adheridos a una interfaz común, localizada en otro archivo Java, que proporciona mayor seguridad, evitando el borrado accidental de argumentos o funciones y permitiendo a otros desarrolladores ver de forma sencilla todos los métodos disponibles en la capa.

Debido a que la estructura interna, tanto a nivel de clases como a nivel conceptual es demasiado compleja como para que el lector pueda entenderla en un documento de esta longitud y con este ámbito, se ha decido no dar detalles concretos sobre la implementación de los métodos descritos en el apartado de diseño.

Sin embargo, conociendo la arquitectura del programa que ha sido descrita y los métodos y sus argumentos, el lector de este trabajo podrá comprender e intuir su implementación.

10 Conclusiones

10.1 Conclusiones generales

Uno de los propósitos principales de un trabajo de fin de grado (TFG) es la de la realización y redacción de un trabajo académico sobre una materia relacionada con su carrera y con sus gustos.

En este sentido, todas mis espectativas se han visto cumplidas. Durante el desarrollo de este trabajo he podido profundizar mis conocimientos en Angular, HTML, SCSS, Java, SQL y Swagger, empleando buenas prácticas, arquitecturas y patrones de diseño. También me he familiarizado con el proceso de gestión de versiones y he colaborado con otros desarrolladores en el entorno de la empresa.

Durante el transcurso de la redacción de este documento he aumentado notablemente mi conocimiento sobre Word para la redacción de trabajos académicos.

Cabe destacar que la concepción y el desarrollo de este trabajo está relacionada con mi estancia en prácticas en ODEC. Esta aplicación será utilizada en un entorno real por centenares de usuarios y en las próximas semanas desde la redacción de este trabajo entrará en fase de pruebas y, posteriormente, en fase de producción.

10.2 Posibles mejoras

Como consecuencia de las limitaciones temporales en las que se ha producido este trabajo, existen algunas mejoras que podrán ser implementadas en el futuro:

- Uso de la librería de detección de bots y prevención de spam y abuso reCAPTCHA. En recientes auditorías a la empresa se detectó la necesidad de proteger las plataformas del uso indebido por parte de *bots* o programas malignos. Su uso no representaría una molestia para el usuario de la aplicación.
- Limpieza de código y reestructuración de componentes que pudieran simplificarse.
- Mejoras de rendimiento y reducción del tamaño de las imágenes y logos empleados.
- Implementación de la carga bajo demanda, para reducir el tamaño inicial de la aplicación *web*.

II
Presupuesto

11 Partidas presupuestarias

En los siguientes apartados se desglosarán los costes del desarrollo de este proyecto.

11.1 Costes humanos

Los costes humanos de este proyecto se corresponden a los sueldos de dos desarrolladores: desarrollador junior y desarrollador senior.

La mayor parte del trabajo ha sido realizada por el alumno, establecido como desarrollador junior. El desarrollador senior se ha encargado de tareas de supervisión y consultoría.

El salario del desarrollador junior se ha obtenido a partir de los datos estadísticos del sector que establecen que el salario medio de un programador junior en España es de en torno a los 19.500€ brutos [10].

El salario del desarrollador senior ha sido establecido de la misma forma en un valor en torno a los 34.000€ brutos [11].

Para el cómputo de horas del desarrollador junior se he establecido el número de horas correspondientes a los créditos ECTS otorgados por la realización de un TFG.

El cómputo de horas del desarrollador senior se corresponde al número de horas aproximadas de sus servicios.

PUESTO	€/H	Nº HORAS	SUBTOTAL
DESARROLLADOR JUNIOR	10,15	300	3.045€
DESARROLLADOR SENIOR	17,70	20	354€
		TOTAL	3399€

Tabla 18: Costes humanos

11.2 Costes materiales

Como costes materiales puede asignarse a este proyecto el ordenador personal con el que se ha hecho el desarrollo, su gasto en electricidad durante ese periodo de tiempo y el gasto en conexión a Internet. El software utilizado no ha tenido ningún coste.

CONCEPTO	COSTE	PERIODO AMORTIZACIÓN (MESES)	PERIODO DE USO (MESES)	SUBTOTAL
ORDENADOR DE ESCRITORIO Y PERIFÉRICOS	2.500€	60	5	210 €
			TOTAL	210€

Tabla 19: Costes materiales

Para el coste de la electricidad y del acceso a Internet se ha optado por utilizar el precio medio a nivel nacional del consumo eléctrico y de Internet del ordenador [12].

Precio de la luz obtenido sin discriminación horaria [13] en base a tarifa One Luz de Endesa.

CONCEPTO	PRECIO	PERIODO	SUBTOTAL
ELECTRICIDAD	0,1998 €/kW/h	300 h	28,75 €
INTERNET	53€/mes	5 meses	265€
		TOTAL	293,75€

Tabla 20: Costes de suministros

11.3 Costes imprevistos

Se ha tomado una partida de costes en previsión de posibles gastos. Dicha partida consta del 5% del coste total del proyecto.

11.4 Impuestos

Se ha establecido como impuesto aplicable el IVA, con valor de 21% de los costes totales del proyecto.

11.5 Presupuesto final

El precio final del proyecto se muestra en la siguiente tabla.

CONCEPTO	IMPORTE (€)
COSTES HUMANOS	3399
COSTES MATERIALES	210
COSTES DE SUMINISTROS	293,75
COSTES IMPREVISTOS	195,14
TOTAL ANTES DE IMPUESTOS	4097,89
IMPUESTOS	860,56
TOTAL	4958,45

Tabla 21: Resumen del presupuesto

Bibliografía

[1] «Devsaran,» [En línea]. Available: https://www.devsaran.com/blog/history-web-application-development. [Último acceso: 20 Agosto 2019].

[2] «Wikipedia - CGI,» [En línea]. Available: https://en.wikipedia.org/wiki/Common_Gateway_Interface. [Último acceso: 2019 Agosto 21].

[3] «Techopedia,» [En línea]. Available: https://www.techopedia.com/definition/4865/common-gateway-interface-cgi-web-development. [Último acceso: 21 Agosto 2019].

[4] «Wikipedia - JQuery,» [En línea]. Available: https://es.wikipedia.org/wiki/JQuery. [Último acceso: 20 Agosto 2019].

[5] «StackOverflow,» [En línea]. Available: https://insights.stackoverflow.com/survey/2019#technology. [Último acceso: 20 Agosto 2019].

[6] «ODEC,» [En línea]. Available: http://www.odec.es/es/. [Último acceso: 21 Agosto 2019].

[7] «TypeScript,» [En línea]. Available: https://www.typescriptlang.org/. [Último acceso: 2019 Agosto 22].

[8] «Netmarketshare,» [En línea]. Available: https://www.netmarketshare.com/browser-market-share.aspx?options=%7B%22filter%22%3A%7B%7D%2C%22dateLabel%22%3A%22Trend%22%2C%22attributes%22%3A%22share%22%2C%22group%22%3A%22browserVersion%22%2C%22sort%22%3A%7B%22share%22%3A-1%7D%2C%22id%22%3A%22browsersD. [Último acceso: 2019 Agosto 23].

[9] «BBVAOpen4U,» [En línea]. Available: https://bbvaopen4u.com/es/actualidad/api-rest-que-es-y-cuales-son-sus-ventajas-en-el-desarrollo-de-proyectos. [Último acceso: 2019 Agosto 28].

[10] «Indeed,» [En línea]. Available: https://www.indeed.es/salaries/Programador/a-junior-Salaries. [Último acceso: 02 Septiembre 2019].

[11] «Indeed,» [En línea]. Available: https://www.indeed.es/salaries/Desarrollador/a-senior-Salaries. [Último acceso: 02 Septiembre 2019].

[12] «Kelisto,» [En línea]. Available: https://www.kelisto.es/internet/actualidad/los-espanoles-pagan-un-27-4-mas-por-internet-que-la-media-europea-6431. [Último acceso: 02 Septiembre 2019].

[13] «Tarifas Gas Luz,» [En línea]. Available: https://tarifasgasluz.com/faq/precio-kwh. [Último acceso: 02 Septiembre 2019].

Anexos

A
Manual del usuario

En este anexo se explicarán los pasos necesarios para la gestión de la contraseña de la cuenta de un usuario de una plataforma de gestión CEMTRIC.

Las gestiones que puede llevar a cabo un usuario son:

- Introducción de la primera contraseña tras la creación de la cuenta.
- Introducción de una nueva contraseña, si los administradores de la plataforma obligan al usuario a cambiar su contraseña por motivos de seguridad.
- Envío de un correo electrónico con un enlace de recuperación de contraseña, en el caso de que el usuario haya olvidado su contraseña anterior.
- Introducción de una nueva contraseña tras redirección desde un correo electrónico de recuperación de contraseña.
- Cambio de contraseña, en el caso en el que el usuario conozca su contraseña anterior y decida cambiarla por motivos de seguridad.

Para realizar las gestiones mencionadas anteriormente se han dispuesto varias páginas cuyo uso será documentado a lo largo del anexo.

12.1 Página de cambio de contraseña

En esta página se puede realizar el cambio de la contraseña del usuario, introduciendo éste su contraseña anterior.

El usuario accede a esta página a través de un enlace recibido en un correo electrónico, pedido bajo demanda en la plataforma de gestión, como el siguiente: *https://www.admin-cem.com/user-management/passwords/change/12345678901234*.

Al acceder a la página, el usuario podrá observar una interfaz como la mostrada en la Figura 32.

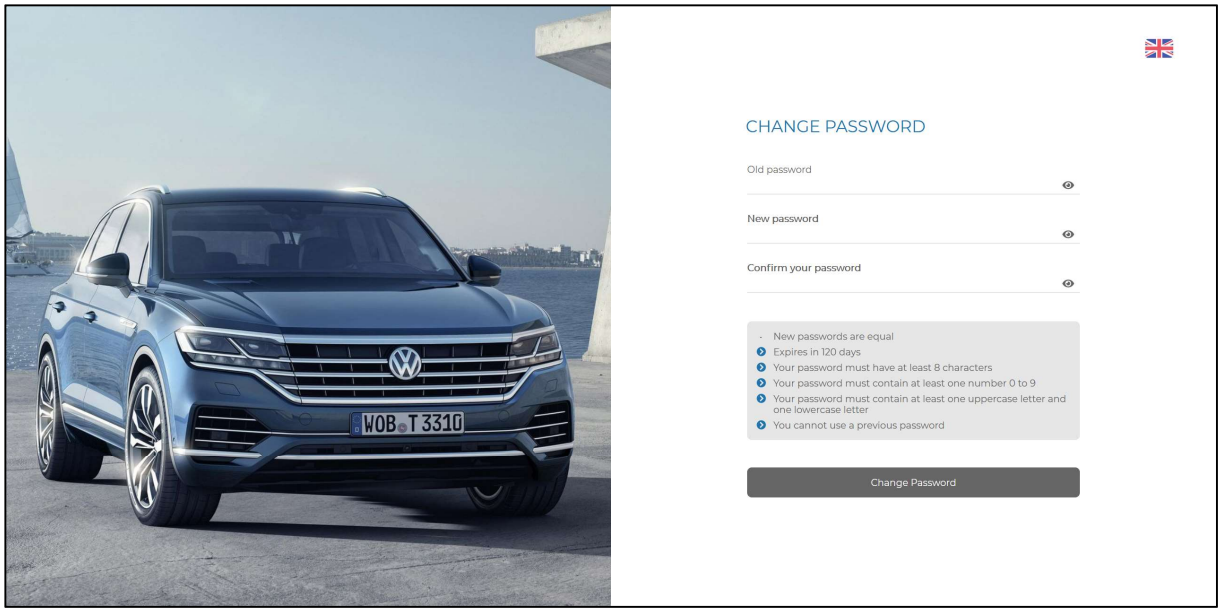

Figura 32: Página de cambio de contraseña

En la esquina superior derecha, puede observarse un botón de cambio de lenguaje. Al ser pulsado da paso a un menú con las banderas de los lenguajes disponibles, mostrado en la . Pulsando alguno de ellos se cambiará el lenguaje de toda la interfaz de la aplicación.

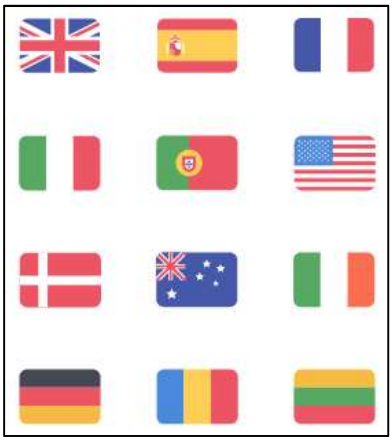

Figura 33: Menú de cambio de lenguaje

En la mitad derecha de la página, hay dispuestas tres entradas de texto o campos. En el primer campo se debe introducir la contraseña anterior del usuario. En el segundo campo se debe introducir la nueva contraseña que el usuario desea establecer para su cuenta y en el tercero debe volver a introducir la contraseña nueva escrita anteriormente. De este modo, se reduce la probabilidad de que el usuario introduzca la contraseña erróneamente.

Más abajo puede encontrarse un área en color gris que contiene las reglas que debe cumplir la contraseña nueva para ser considerada válida. También muestra los mensajes de error en el caso de que no pueda cambiarse la contraseña.

Por último se halla el botón de envío. Este botón permanece bloqueado (tanto estética como funcionalmente) a la pulsación hasta que los tres campos del formulario hayan sido rellenados y coincidan las contraseñas nuevas.

Cuando el usuario pulsa el botón, aparece una animación de carga en el mismo, indicando al usuario que se está procesando su gestión. Si la contraseña nueva es incorrecta, aparecerá en pantalla la regla que ha incumplido. Cuando la contraseña nueva cumpla todas las reglas, aparecerá un mensaje en pantalla que le indicará que la gestión se ha producido con éxito y mostrará un botón que le permita volver a la plataforma CEMTRIC.

12.2 Página de introducción de contraseña

En esta página se pueden realizar las siguientes gestiones:

- Introducción de la primera contraseña tras la creación de la cuenta.
- Introducción de una nueva contraseña, si los administradores de la plataforma obligan al usuario a cambiar su contraseña por motivos de seguridad.
- Introducción de una nueva contraseña tras redirección desde un correo electrónico de recuperación de contraseña.

El usuario accede a esta página a través de un enlace recibido en un correo electrónico como el siguiente: *https://www.admin-cem.com/user-management/passwords/new/12345678901234*.

Al acceder a la página, el usuario visualizará la interfaz mostrada en la Figura 34.

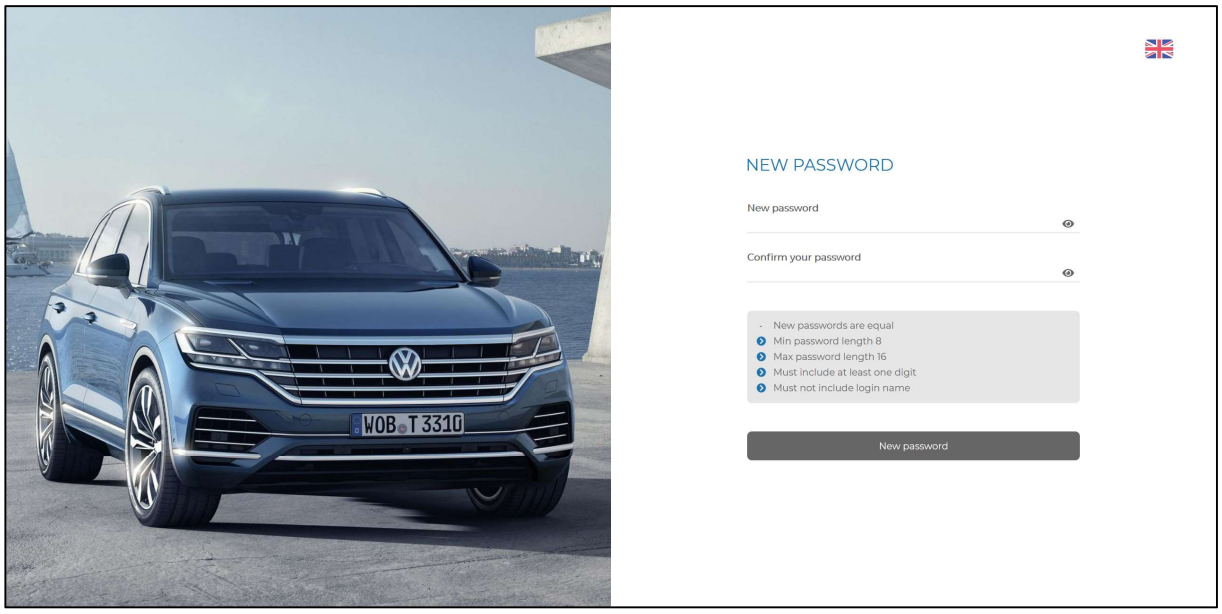

Figura 34: Página de introducción de contraseña

En la esquina superior derecha, puede observarse un botón de cambio de lenguaje. Al ser pulsado da paso a un menú con las banderas de los lenguajes disponibles, como puede verse en la Figura 33. Pulsando alguno de ellos se cambiará el lenguaje de toda la interfaz de la aplicación.

En la mitad derecha de la página, hay dispuestos dos campos. En el primer campo se debe introducir la nueva contraseña que el usuario desea establecer para su cuenta y en el segundo debe volver a introducir la contraseña escrita anteriormente. De este modo, se reduce la probabilidad de que el usuario introduzca la contraseña erróneamente.

Más abajo puede encontrarse un área en color gris que contiene las reglas que debe cumplir la contraseña para ser considerada válida. También muestra los mensajes de error en el caso de que no pueda establecerse la contraseña.

Por último se halla el botón de envío. Este botón permanece bloqueado (tanto estética como funcionalmente) a la pulsación hasta que los dos campos del formulario hayan sido rellenados y coincidan las contraseñas.

Cuando el usuario pulsa el botón, aparece una animación de carga en el mismo, indicando al usuario que se está procesando su gestión. Si la contraseña es incorrecta aparecerá en pantalla la regla que ha incumplido. Cuando la contraseña introducida cumpla todas las reglas aparecerá un mensaje en pantalla que le indicará que la gestión se ha producido con éxito y mostrará un botón que le permita volver a la plataforma CEMTRIC.

12.3 Página de recuperación de contraseña

En esta página se puede realizar la petición de un correo electrónico de recuperación de contraseña.

El usuario accede a esta página a través de un enlace situado en la página de inicio de sesión de la plataforma de gestión correspondiente, indicado a continuación.

Figura 35: Enlace de recuperación de contraseña

Tras pulsar en enlace, el usuario accede a la página mostrada en la Figura 36.

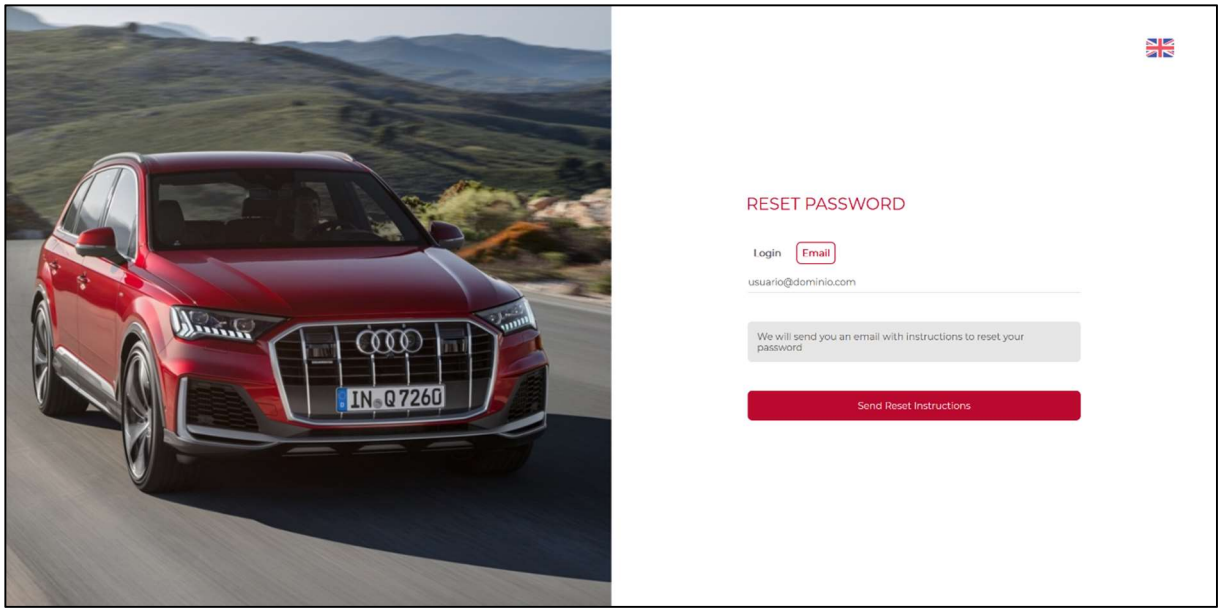

Figura 36: Página de recuperación de contraseña

En la esquina superior derecha, puede observarse un botón de cambio de lenguaje. Al ser pulsado da paso a un menú con las banderas de los lenguajes disponibles, como puede verse en la Figura 33. Pulsando alguno de ellos se cambiará el lenguaje de toda la interfaz de la aplicación.

En la mitad derecha de la página, puede localizarse un selector que permite elegir el tipo de identifiación del usuario que se utilizará para enviarle un correo electrónico, su nombre de usuario o su dirección de correo electrónico. Seleccionada una de las dos opciones, se debe introducir en el único campo disponible el nombre de usuario o la dirección de correo electrónico.

A continuación, puede encontrarse un área en color gris que contiene indicaciones para realizar el proceso con éxito. También muestra los mensajes de error en el caso de que no pueda realizarse el envío del correo electrónico al usuario.

Por último se halla el botón de envío. Este botón permanece bloqueado (tanto estética como funcionalmente) a la pulsación hasta se haya rellenado el campo con el identificador de cuenta.

Cuando el usuario pulsa el botón, aparece una animación de carga en el mismo, indicando al usuario que se está procesando su gestión. Cuando se haya producido el envío del correo electrónico aparecerá un mensaje en pantalla que le indicará que la gestión se ha producido con éxito y mostrará un botón que le permita volver a la plataforma CEMTRIC.

12.4 Páginas de alerta

Las páginas de alerta indican que cualquiera de los procesos descritos anteriormente no pueden comenzar o continuar correctamente.

El usuario accede a estas páginas a través de un redireccionamiento por parte de la aplicación.

Al ser redireccionado a esta página, el usuario puede observar una página similar o igual a la mostrada en la Figura 37.

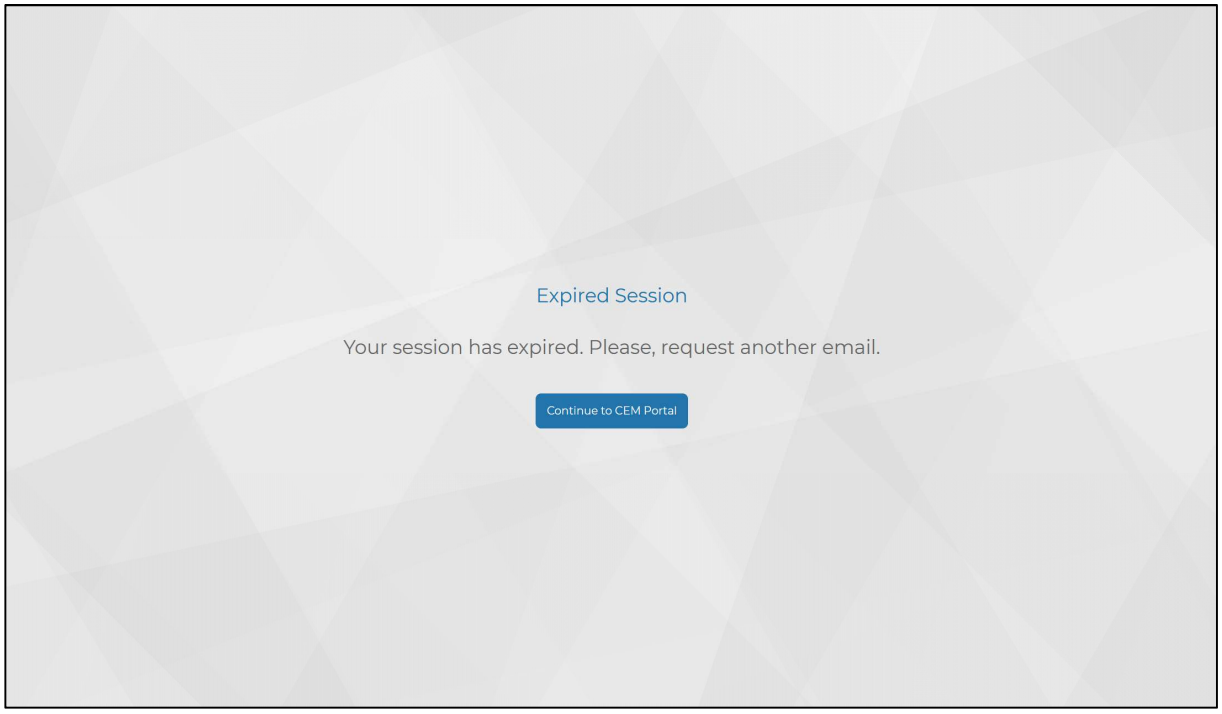

Figura 37: Página de alerta

En orden descendente puede encontrarse el título de la página, que indica el motivo principal del fallo que impide al usuario realizar el proceso. A continuación se halla un párrafo con detalles acerca del fallo. Por último, se halla un botón que redirige al usuario a la plataforma de gestión cuando es pulsado.

El contenido del título y de la descripción varía dependiendo del tipo de fallo que se ha producido. La página a la que redirige el botón depende de la plataforma en la que esté instalada la aplicación.