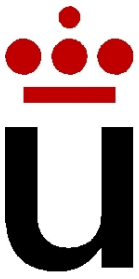


Universidad Rey Juan Carlos

Grado en Ingeniería en Sistemas Audiovisuales y Multimedia

Escuela Técnica Superior de Ingeniería de Telecomunicación

Curso académico 2016-2017

Trabajo Fin de Grado

BACKEND DE UNA APLICACIÓN WEB PARA LA GESTIÓN DE PARTES DE OBRA

Autor: Ignacio Arranz Águeda
Tutor: José María Cañas Plaza

Madrid 2017

A mi familia
y a Patri,
por aguantarme todos estos años.

Índice general

Índice de figuras

Índice de cuadros

Índice de Fragmentos

Capítulo 1

Introducción

En este capítulo se introducen los conceptos de aplicaciones web y sus características así como la motivación principal que ha impulsado el desarrollo de este trabajo de fin de grado. Este TFG se ha desarrollado dentro de un proyecto industrial que aborda el desarrollo de una aplicación web para la gestión y generación de reportes o '*Partes de Obra*' para una empresa constructora de túneles.

1.1. Aplicaciones Web

El desarrollo de aplicaciones web ha evolucionado enormemente en la última década, tanto desde el punto de vista del desarrollo de software (numerosos entornos de desarrollo, bibliotecas, aplicaciones configurables, plugins, etc) y de la administración de sistemas (servicios de alojamiento, técnicas de estabilidad, monitorización, etc) como en funcionalidad.

En los últimos años se ha incrementado el uso de tecnologías web en muchos ámbitos. Se han ido adaptando y transformado para pasar de ser una página web de consulta de información, denominadas páginas web de contenido *estático*, a tener una fuerte interacción con el usuario, personalizándose la información e interactuando con el servidor, mostrando resultados en función de esa interacción del historial, etc, (webs *dinámicas*). En este camino es donde se ha encontrado un punto en común entre las páginas web y los programas, denominándose '*Aplicaciones web*'.

En la ingeniería de software se denomina '`Aplicación web`' a aquellas herramientas que los usuarios pueden utilizar accediendo a un servidor web a través de Internet o de una intranet mediante un navegador, utilizando como protocolo de comunicación `HTTP`. En otras palabras, es una aplicación software que se codifica en un lenguaje soportado por los navegadores en la que se confía la ejecución al navegador y a un servidor web. El modelo que utilizan las aplicaciones web es típicamente el modelo *cliente-servidor*.

Las aplicaciones web son populares debido a lo práctico del navegador web como cliente ligero, a la independencia del sistema operativo, así como a la facilidad para actualizar y mantener aplicaciones web sin distribuir e instalar software a miles de usuarios potenciales.

Las aplicaciones web quedan estructuradas en dos partes:

- *Cliente:* Permite crear interfaces de usuario atractivos e intuitivos para facilitar la introducción de datos por parte del usuario y permite la comunicación con el

servidor. Las aplicaciones web que se basan en una mucha interactividad con el usuario suelen hacer especial énfasis en esta parte, ya que al ejecutarse dentro del navegador, hacen esta interacción muy rápida y fluida.

- *Servidor:* Es el encargado de recibir las peticiones, principalmente `HTTP` aunque también puede soportar otras tecnologías como `AJAX` o `WebSockets`, y responder de manera adecuada a estas peticiones. Permite implementar el comportamiento y la lógica de la aplicación, recibir peticiones, realizar búsquedas, etc, son algunos de los ejemplos que suelen estar ligados a una base de datos que nutre a la aplicación de todo el contenido.

El concepto de aplicación web se remonta al primer lenguaje de programación dedicado para este ámbito, `Perl`, inventado por Larry Wall en 1987. Desde entonces hasta hoy en día, los navegadores han ido ganando en rapidez, capacidad de cómputo y funcionalidades. Los estándares y lenguajes de programación dedicados a este sector han alcanzado un alto grado de optimización y madurez que permite ejecutar aplicaciones en cualquier dispositivo como si de una aplicación de escritorio se tratara usando únicamente el navegador. Las figuras 1.1, 1.2 1.3 y 1.4 son un ejemplo de aplicación web.

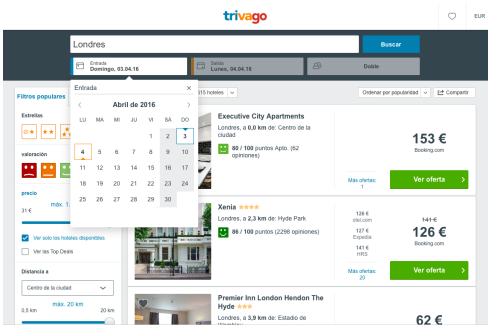


Figura 1.1: Trivago

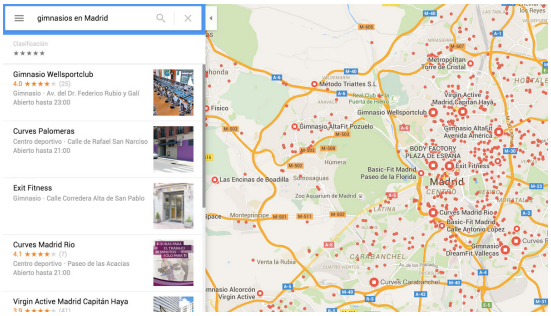


Figura 1.2: Google Maps

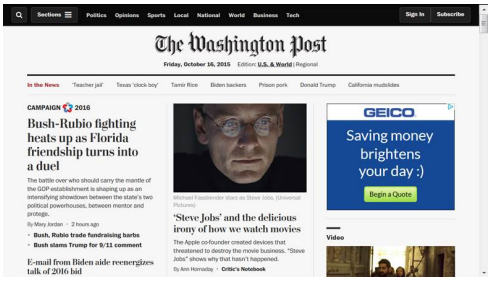


Figura 1.3: Washigton Post

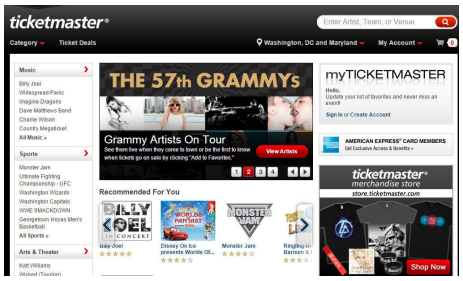


Figura 1.4: TicketMaster

Gran parte del crecimiento del uso de las aplicaciones web reside en que `JavaScript` se ha convertido en el lenguaje estándar para dotar al cliente de la lógica necesaria. Se puede observar en la figura 1.5 cómo el rendimiento del intérprete de `JavaScript` ha crecido exponencialmente con los años. Esto permite que las aplicaciones web aumenten la carga computacional en el cliente, siendo capaces de resolverlo de un modo suficientemente ágil. Esta evolución deja clara la apuesta que se está haciendo por el uso de aplicaciones web en el sector.

Hoy por hoy las aplicaciones web son usadas constantemente para todo tipo de entornos, ya sea comprar por Internet, consultar prensa personalizada, multitud de herramientas

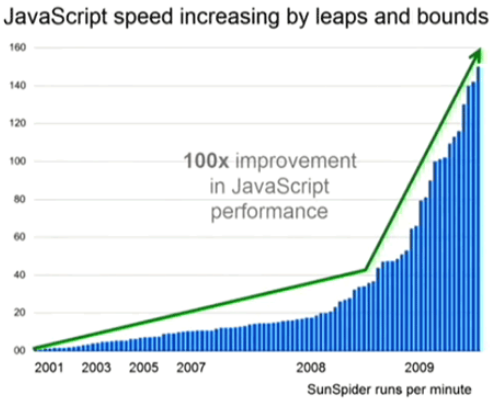


Figura 1.5: Evolución del rendimiento de JavaScript

de trabajo, acceso a contenido multimedia, llamadas de voz, vídeo conferencias, etc.

Un ejemplo claro de aplicación web puede ser la página `Amazon`[1] (figura 1.6) que se basa en una de las características más importantes de las aplicaciones web, el contenido dinámico. Esta característica nutre a la página de un contenido personalizado para cada usuario, ofreciéndole productos en función de compras anteriores o lugares que ha visitado.

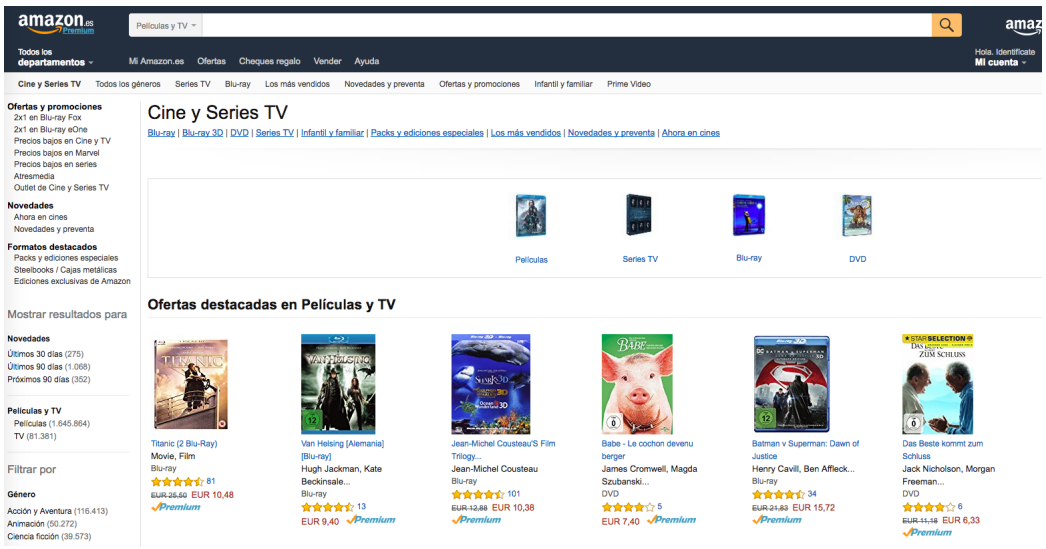


Figura 1.6: Ejemplo de Amazon como página dinámica

Otro ejemplo muy popular es `Netflix` (figura 1.7), una de las plataformas de vídeo en streaming más popular que existe en la actualidad. Es otra buena muestra de aplicación web basada en el contenido dinámico, recomendando series o películas relacionadas con las que ya han sido reproducidas, todas ellas bajo demanda, y con posibilidad de descarga para su visualización sin acceso a internet. `Netflix` y otras plataformas similares son la demostración del crecimiento en los últimos años de las aplicaciones web en el ámbito del contenido multimedia.
El uso de aplicaciones web frente a aplicaciones tradicionales de escritorio ofrece unas ventajas e inconvenientes que se detallan en los puntos 1.1.1 y 1.1.2.

[1] `https:www.amazon.es`

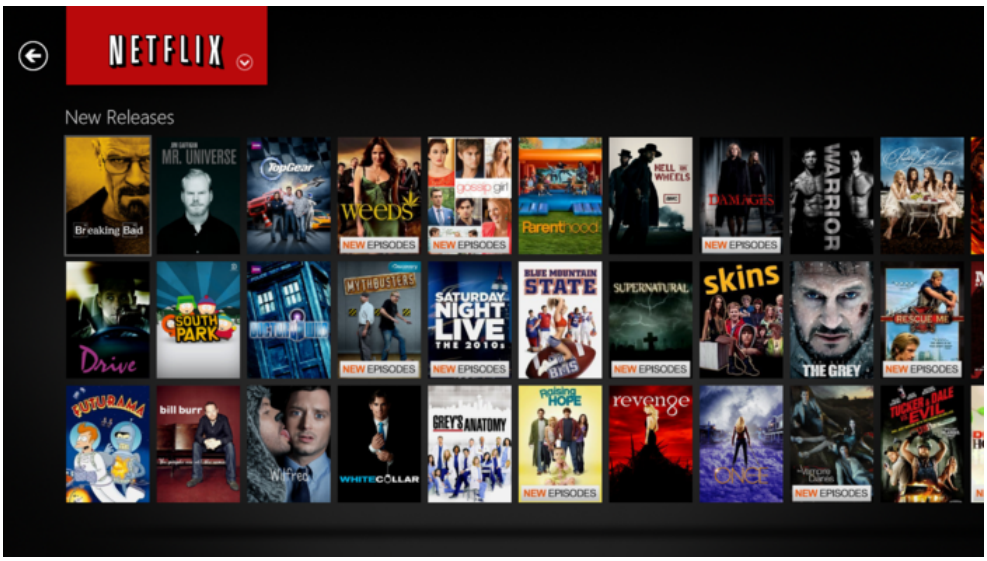


Figura 1.7: Ejemplo de Netflix como aplicación web dinámica

1.1.1. Ventajas

Los aspectos positivos del uso de una aplicación web podemos desglosarlos en los siguientes puntos:

- El cliente sólo necesita tener acceso a un navegador web con conexión a internet para usar la aplicación, por lo que no necesita descargar, instalar ni configurar software específico.
- Son multiplataforma. Puesto que el único software necesario en el lado del cliente es un navegador web, podemos usar una aplicación web desde cualquier dispositivo que disponga de uno. Además, desde el punto de vista de la programación, simplifica mucho el proceso ya que sólo es necesario desarrollar un único software para todas las plataformas (`Linux`, `MS-WindowsMacOS`) y no uno para cada una, traduciéndose en un menor coste de desarrollo y, por tanto, económico.
- Hacen uso de la responsividad. El acceso a la aplicación desde el navegador web permite que el contenido visual de la página se adapte a la pantalla del dispositivo que está accediendo. Igual que en el punto anterior, con un único desarrollo se estructura cada uno de los elementos de los que se compone la página web. Puede verse en la figura 1.8 un ejemplo.

Figura 1.8: Adaptación de la estructura mediante responsividad

- No requieren actualizaciones por parte del cliente. Una actualización por parte del servidor sobre la aplicación provoca que el acceso de un cliente descargue el nuevo

contenido. Este proceso es ajeno al cliente ya que no tiene que realizar ningún procedimiento para disponer de la última versión.

- No hay problemas de compatibilidad entre versiones pues todos los usuarios trabajan con la última versión lanzada en el servidor.
- Menores requisitos de memoria. Las aplicaciones web suelen tener menor demanda de recursos *hardware* que el software instalado localmente.

1.1.2. Inconvenientes

Como contrapartida, las aplicaciones web presentan las siguientes desventajas:

- Requieren conexión a la red. Aunque cada vez se está popularizando más el uso de las aplicaciones web de manera *offline*, sí se requiere, al menos, un primer acceso para una descarga previa de los documentos necesarios para una navegación sin conexión. Por tanto, son dependientes del tipo de conexión a la red así como posibles problemas de acceso al servidor (p.e. saturación). Un fallo en uno o varios servidores podría dejar completamente sin servicio a las aplicaciones.
- Suelen tener menor rendimiento que las aplicaciónes de escritorio, ya que no son tan eficientes utilizando los recursos *hardware* del lado cliente.
- El usuario no puede elegir la versión de la aplicación que desea utilizar, ya que sólo puede ser usada aquella alojada en el servidor. Esta desventaja no suele ser demasiado significativa excepto en casos muy concretos.
- Son menos seguras que las aplicaciones de escritorio puesto que muchas operaciones necesitan de transacción con el servidor.
- El acceso a los datos es más lento debido a que el contenido es descargado cuando se accede al servidor. En el caso de las aplicaciones de escritorio, al estar alojadas en la máquina, el acceso es inmediato.

Actualmente las aplicaciones web están viviendo un auge en su uso promovido por todas las nuevas novedades que se han ido introduciendo en las tecnologías web.

1.2. Tecnologías Web

En este apartado se repasan las tecnologías web más comunes actualmente cuando se desarrollan aplicaciones web, tanto en el lado cliente como en el lado servidor, además de la comunicación entre cada una de las partes.

1.2.1. HTTP

Hypertext Transfer Protocol o HTTP (protocolo de transferencia de hipertexto) es el protocolo de comunicación, basado en texto, que permite las transferencias de información en la World Wide Web. `HTTP` fue desarrollado por el *World Wide Web Consortium* y la *Internet Engineering Task Force*, colaboración que culminó en 1999 con la publicación de una serie de RFC, el más importante de ellos es el RFC 2616 que especifica la versión 1.1 de este protocolo. `HTTP` define la sintaxis y la semántica que utilizan los elementos de

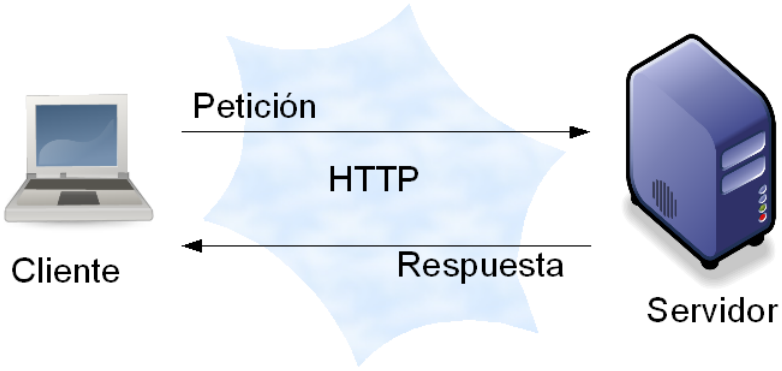


Figura 1.9: Modelo Cliente-Servidor

software de la arquitectura web (clientes, servidores, proxies) para comunicarse. Trabaja sobre el protocolo TCP de la capa de transporte.

Es un protocolo orientado a transacciones y sigue el esquema petición-respuesta entre un cliente y un servidor. Al cliente que efectúa la petición (un navegador web) se lo conoce como agente del usuario. A la información transmitida se le llama recurso y se identifica mediante un localizador uniforme de recursos (URL). El recurso es el resultado de la ejecución de un programa en el lado servidor como puede ser: una consulta a una base de datos, solicitar un documento, etc.

Además `HTTP` es un protocolo sin estados, por lo que no guarda información sobre conexiones anteriores. Actualmente el desarrollo de aplicaciones web requiere frecuentemente guardar información sobre estado. Para este cometido se utilizan las *Cookies* que es información de un servidor que puede almacenar en el navegador del cliente durante un tiempo limitado. Esta herramienta se utiliza con múltiples propósitos como, por ejemplo, implementar la noción de sesión en aplicaciones web.

1.2.2. Tecnologías Cliente

Hasta hace unos años, las aplicaciones web no podían competir con las aplicaciones de escritorio por los motivos vistos en la sub-sección 1.1.2, pero en últimos tiempos las innovaciones en las tecnologías web han permitido a las aplicaciones web igualarlas en muchos aspectos. Algunas de estas innovaciones son: el almacenamiento local, los "`Web Workers`", la introducción de gráficos web acelerados por hardware GPU en navegadores web, funcionalidad *offline*, `AJAX`, etc.

El modelo más usado para la construcción de páginas web en el lado cliente es `HTML`. `HTML`, en su versión 5, es un lenguaje de marcado estandarizado para la elaboración de páginas web, que permite definir su estructura y contenido.

A lo largo de sus diferentes versiones, se han incorporado y suprimido diversas características, con el fin de hacerlo más eficiente y facilitar el desarrollo de páginas web compatibles con distintos navegadores y plataformas (PC de escritorio, portátiles, teléfonos inteligentes, tabletas, etc.). No obstante, para interpretar correctamente una nueva versión de `HTML`, los desarrolladores de navegadores web deben incorporar estos

cambios. Por estas razones, aún existen diferencias entre distintos navegadores y versiones al interpretar una misma página web.

El fuerte uso del estándar está haciendo que, claramente, se apueste por `HTML5`, reduciendo estas diferencias entre navegadores y dejando atrás otros entornos muy conocidos y no estandarizados como `Adobe Flash Player`, perteneciente a `Adobe`.

`HTML5` gestiona mejor los recursos del dispositivo donde se está renderizando la página por lo que disminuye los tiempos de carga, haciéndo más fluida la navegación. Además permite el acceso a elementos del equipo que hasta ahora requerían de *plugins* externos. Estos pueden ser: acceso a la cámara o micrófono del ordenador para permitir efectuar llamadas o video conferencias, renderización de gráficos en tiempo real, como se ve en la figura 1.10, usando el hardware del equipo. Esto permite que aplicaciones con gran carga de procesamiento gráfico aproveche los recursos *hardware* locales y ofrezca resultados equiparables a aplicaciones que requieren de instalación.

Figura 1.10: Gráficos generados mediante JavaScript

Dado que `HTML5` aporta únicamente la estructura, son necesarios otros elementos que le aporten el estilo y presentación visual que se busca en la construcción de páginas web. Es por ello que, al igual que ha ocurrido con `HTML` y su evolución, se opte por `CSS`, en su versión 3, para añadir los estilos.

Estos dos lenguajes están estandarizados por el *World Wide Web Consortium* o `W3C`. Es un consorcio internacional que genera recomendaciones y estándares en el ámbito de las tecnologías web. Los estándares han ayudado a que se cree una uniformidad en el uso de estas tecnologías que ha permitido el crecimiento.

Por último, en el lado cliente se utiliza el lenguaje `JavaScript` para la interacción con el usuario y añadir lógica y dinamismo a las páginas web. El uso más común de `JavaScript` es escribir funciones incluidas en páginas `HTML`. Algunos ejemplos sencillos de este uso son:

- Cargar nuevo contenido para la página o enviar datos al servidor a través de una

comunicación asíncrona con el servidor (AJAX) sin necesidad de recargar la página completa.

- Animación de los elementos de página, hacerlos desaparecer, cambiar su tamaño, moverlos, etc.
- Contenido interactivo como juegos y reproducción de audio y vídeo, etc.
- Validación de los valores de entrada de un formulario web para asegurarse de que son aceptables antes de ser enviado al servidor.

Figura 1.11: Estándar Web

Estos tres lenguajes estandarizados hacen que la tendencia al uso de aplicaciones web se vea incrementada notablemente. Los resultados finales cada vez son menos distantes con una aplicación de escritorio en cuanto recursos, tiempos de carga y usabilidad.

1.2.3. Tecnologías Servidor

En el otro lado de las aplicaciones web se encuentra el servidor. Es el encargado de recibir las solicitudes HTTP que llegan desde el cliente y responder a ellas de manera adecuada. A diferencia del cliente, las tecnologías usadas en el servidor son más heterogéneas. El lado servidor no cuenta con una estandarización que provoque una tendencia de uso de un tipo u otro de tecnología debido a que existe más de un paradigma, multitud de lenguajes de programación, bases de datos, sistemas operativos, etc. Es en este contexto donde han ido apareciendo distintas formas de desarrollar un servidor web, denominándose *entornos de desarrollo*.

Actualmente existen varios entornos del lado del servidor muy populares entre la comunidad de desarrolladores. Entre ellos se encuentra Node.js[2], un entorno en tiempo de ejecución multiplataforma, de código abierto, basado en el lenguaje de programación interpretado ECMAScript o (JavaScript), asíncrono y con una arquitectura orientada a eventos.

Figura 1.12: Entorno de desarrollo Node.js

[2] https://nodejs.org/es/

Otro de los entornos de desarrollo en el lado servidor más populares basado en el lenguaje de programación `Python` es `Django`[3]. Es un entorno web de código abierto que respeta el patrón de diseño conocido como Modelo–Vista–Controlador. La meta fundamental de `Django` es facilitar la creación de sitios web complejos. Pone énfasis en el re-uso, la conectividad y extensibilidad de componentes, el desarrollo rápido y el principio "No te repitas"(DRY, del inglés *Don't Repeat Yourself*). `Python` es usado en todas las partes del entorno, incluso en configuraciones, archivos y en los modelos de datos.

Figura 1.13: Entorno de desarrollo Django

Otro ejemplo, con una filosofía de diseño muy parecida a la de `Django` es `Ruby on Rails`[4]. También conocido como RoR, es un entorno de código abierto escrito en el lenguaje de programación `Ruby`, siguiendo también el paradigma del patrón Modelo-Vista-Controlador (MVC). Trata de combinar la simplicidad con la posibilidad de desarrollar aplicaciones del mundo real escribiendo menos código que con otros entornos y con un mínimo de configuración. El lenguaje de programación Ruby hace uso de una sintaxis que muchos de sus usuarios encuentran muy legible.

Figura 1.14: Entorno de desarrollo Ruby on Rails

Cada uno de estos entornos tendrá con toda probabilidad una base de datos asociada a los modelos definidos. Una de las grandes ventajas de estos entornos de desarrollo descritos es la facilidad con la que se conectan a las bases de datos, ya sean relacionales o no relacionales. La diferencia entre estas dos configuraciones es la manera en la que se estructuran los datos. Las bases de datos relacionales (como por ejemplo `MySQL`, `PostgreSQL` o `Microsoft SQL Server`) siguen un esquema de tablas donde el número de columnas es fijo para cada usuario. Por el contrario, las no relacionales (`MongoDB` como la más popular) aportan mayor dinamismo y pueden contener diversidad de campos para un mismo usuario.

La existencia de una base de datos para las aplicaciones web tiene una gran importancia.

[3] `https://www.djangoproject.com/`
[4] `http://rubyonrails.org.es//`

Para los ejemplos de aplicaciones presentadas en las sección 1.1 es necesaria una base de datos que almacene todo tipo de características del usuario para poder ofrecerle después recomendaciones basándose en búsquedas o contenido visitado, almacenar pedidos acerca de compras realizadas, etc.

Los entornos aquí presentados son sólo un ejemplo ilustrativo, no exhaustivo, cada uno con unas características y particularidades que determinarán la elección de uno u otro a la hora de desarrollar una aplicación web. Otros entornos con otros lenguajes de programación como pueden ser `Java`, y su entorno `Spring`, o uno de los lenguajes orientado a servidor más popular, `PHP`, resuelven también el desarrollo de una aplicación web.

1.2.4. Servidores en producción

Los entornos presentados en la sección 1.2.3 suelen ir acompañados de otros servidores web utilizados para desplegar y alojar los sitios en vivo o aplicaciones web desarrolladas, denominados *Servidores en producción*. Estos se encargan de gestionar las peticiones `HTTP` que el cliente solicita y encaminarlas hacia la aplicación web desarrollada. Los servidores en producción más populares son `Apache` y `Nginx`. El primero de ellos abarca más de la mitad de los sitios web activos en la red ya que es un servidor web robusto y flexible. Por otro lado, `Nginx` está ganando popularidad dentro del mercado pese a llevar menos años activo.

1.3. Aplicación web de partes de obra para una empresa.

Este TFG se ha desarrollado en el contexto de un proyecto para una empresa que tiene el objetivo de digitalizar un proceso de introducción de *Partes de Obra*. El procedimiento para la creación de estos partes se basa en la introducción de datos ofreciendo un resumen diario de la jornada con una granularidad de introducción por hora. Hasta hace poco, se creaba un documento `Excel` con varias columnas que era impreso en papel y se distribuía a los operarios. Una vez relleno en papel pasaba por un proceso de escaneado y envío a la oficina central donde volvía a ser pasado, de nuevo, al documento de tipo `Excel`. Este proceso lleva consigo un alto coste en tiempo y no da la posibilidad de acceder a los datos inmediatamente ni permite hacerlo desde cualquier ubicación geográfica.

La principal motivación de la empresa es la de digitalizar este procedimiento buscando que sea más rápido y sencillo. Además, al guardar la información de cada parte de obra en una base de datos remota, permite la consulta y edición de estos de manera instantánea y muy sencilla desde prácticamente cualquier dispositivo electónico que cuente con un navegador web.

Para llevar a cabo este digitalización se ha optado por una solución con tecnologías web. Una de las principales ventajas de estas tecnologías radica en la simplicidad pero, a su vez, mayor eficiencia en la recolección de información en un procedimiento que hasta ahora no tenía carácter inmediato ni acceso a los datos remotamente. La inclusión en este proyecto de una base de datos que refleja el material y personal desplegado en una obra, aporta mayor control acerca de la distribución de cada elemento. Para la introducción de datos en esa base de datos es donde entra en juego la aplicación web de este TFG. De esta manera, un operario puede introducir desde el navegador web de su dispositivo

electrónico (teléfono inteligente, tablet, ordenador portátil, etc) los datos del formulario, incluso aunque no haya conexión a Internet y cuando exista, éstos se enviarán al servidor y serán tratados. Todas las transacciones se almacenarán en la base de datos de manera instantánea y su visualización en otra parte geográfica (y con cualquier dispositivo) será posible dado que accederán al mismo portal al ser multiplataforma.

En el capítulo 2 se verán los objetivos concretos tanto de la aplicación como de este TFG así como la metodología seguída y plan de trabajo.En el capítulo 3 se mostrará un resumen de las tecnologías usadas. A continuación, se procederá a explicar en detalle el desarrollo y funcionamiento del servidor en el capítulo 4. Después, se motrarán los experimentos realizados en el capítulo5 para asegurar la consistencia del servidor. Por último, un capítulo de conclusiones donde se expondrán los objetivos cumplidos así como los trabajos futuros que se plantean.

Capítulo 2

Objetivos

Una vez se ha introducido el contexto en el que se realiza este TFG, en este capítulo se presentan los objetivos y requisitos que se pretenden alcanzar, aplicada una planificación sobre el proyecto y la metodología concreta hasta llegar a la solución final.

2.1. Marco de la empresa

Este TFG forma parte de un proyecto industrial completo en el que se ha colaborado con una empresa. El objetivo de la empresa es agilizar un procedimiento interno de introducción de reportes diarios con herramientas digitales. La introducción de estos reportes o "partes de obra", se venía realizando hasta entonces usando papel físico que era escaneado y enviado para que, en su recepción, se digitalizara al formato excel y posterior almacenamiento, tanto física como digitalmente.

El proyecto completo de digitalización resuelve el problema de los tiempos y la inmediatez en el que los datos se hacen efectivos en la central de la empresa, además de ser un portal donde se concentra información acerca del personal, maquinaria y proyectos existentes. Se propone una solución, haciendo uso de las tecnologías web, que ofrece mejor soporte para la introducción, almacenamiento, comprobación y descarga de elementos de manera simple y casi instantánea para el usuario. Mediante el uso de tecnologías web se pretende solucionar la inmediatez en la que los datos son accesibles además del acceso a la aplicación web desde cualquier punto con conexión a la red.

El uso del navegador web como interfaz y las herramientas software multiplataforma, facilitan el desarrollo ya que, con un único programa escrito, la aplicación puede ser usada en todos los sistemas operativos que cuenten con un navegador web que sea compatible con los estándares HTML5, CSS3 y JavaScript.

2.2. Objetivo del TFG

El TFG se enmarca en el proceso de introducción de Partes de Obra facilitando la introducción y consulta de los datos. Dentro de este marco, el objetivo principal de este TFG es la elaboración del lado servidor (*backend*) de una herramienta software que permita una mayor productividad en el proceso de elaboración de un parte de reporte diario. Con esta herramienta, cada uno de estos partes quedará disponible para cada usuario que quiera acceder a ellos o descargarlos de manera instantánea y sin ninguna exigencia de sistema operativo o aplicación de escritorio, simplemente con el uso del

navegador web disponible por defecto en cada uno de los dispositivos.

Este objetivo principal se ha dividido en cuatro subobjetivos que tratan cada una de las partes de la aplicación web:

- Primero, el modelado de datos que refleje la lógica necesaria para la aplicación web de partes de obra y materizalizarlo en una base de datos relacional.
- Segundo, el desarrollo de un servidor web capaz de recibir partes de obra y almacenarlos en bases de datos para que luego puedan ser descargados o consultados en varios formatos.
- Tercero, desarrollar funcionalidades avanzadas como son: la exportación en formato `Excel` y `pdf` de los Partes de Obra, materializar un sistema web de notificaciones que permita al servidor avisar proactivamente a los clientes de cambios interesantes y el acceso a la aplicación con soporte multilenguaje mediante herramientas de internacionalización.
- Y cuarto subobjetivo, integrar la aplicación web en producción con servidores dedicados para tal fin y que cuenten con módulos de seguridad para materializar transacciones seguras.

2.3. Requisitos

Además, la aplicación web debe satisfacer una serie de características adicionales que se resumen en los siguientes puntos:

- *Uso de tecnologías de vanguardia:* Se empleará software asentado y estandard de facto, por su fuerte comunidad de apoyo y que está siendo usado por multitud de aplicaciones para un desempeño similar al que aquí se presenta.
- *Acceso por roles:* Se necesita una aplicación que se adapte a cada usuario y a cada petición en función de los privilegios asignados, por lo que la información será seleccionada para cada uno de ellos.
- *Guardado y actualización en tiempo real:* La inmediatez de los datos tanto en su almacenamiento como en su disposición es otro elemento de gran importancia.
- *Intuitiva:* La aplicación será usada por distintos usuarios y con distintos tipos de conocimientos a nivel de software. Que la herramienta sea de fácil uso para cada uno de ellos es un requisito fundamental.

2.4. Metodología

Para este TFG se ha empleado el método de *desarrollo en espiral*, muy usado en ingeniería de software. Permite establecer pequeños objetivos para cada una de la iteraciones, formado por un conjunto de actividades. Estas actividades consisten en establecer unos *objetivos*, planificados previamente en función de los cumplidos en la semana anterior. *Estructurados* por prioridad, se *estudia* su diseño y se *programa* una solución. Una vez resuelto se procede a *pruebas* de los componentes y se *evalúan los resultados y objetivos* marcados de cara a proponer unos nuevos, comenzando una nueva iteración en el bucle. En la figura 2.1, se muestra un diagrama con la estructura de la metodología seguida durante el desarrollo del proyecto.

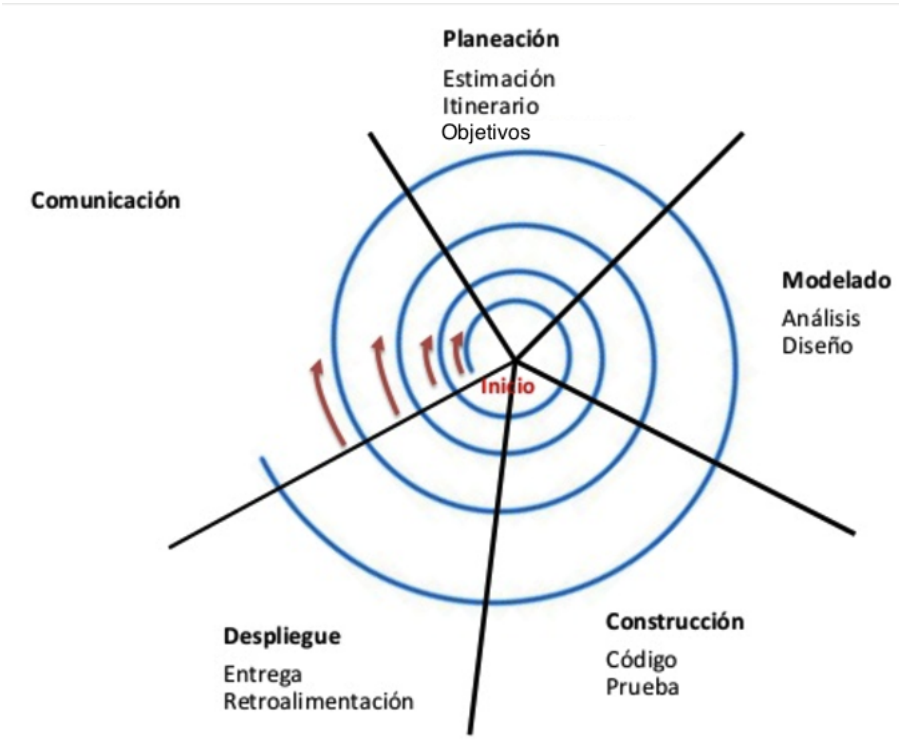


Figura 2.1: Modelo en espiral

Comenzando con prototipos pequeños de la aplicación, en cada una de las iteraciones va perfilándose y aumentando los componentes y funcionalidad hasta llegar a la solución final.

Para este proyecto se ha usado, además, la herramienta de control de versiones `GitLab`, perteneciente a `Git`, que permite un seguimiento del estado del proyecto así como de cada uno de los archivos. Incluyendo un centro de incidencias, permite tener los puntos de desarrollo con un seguimiento más profundo. El repositorio en esta plataforma está alojado de manera privada. Puede verse el resumen del proyecto en la figura 2.2.

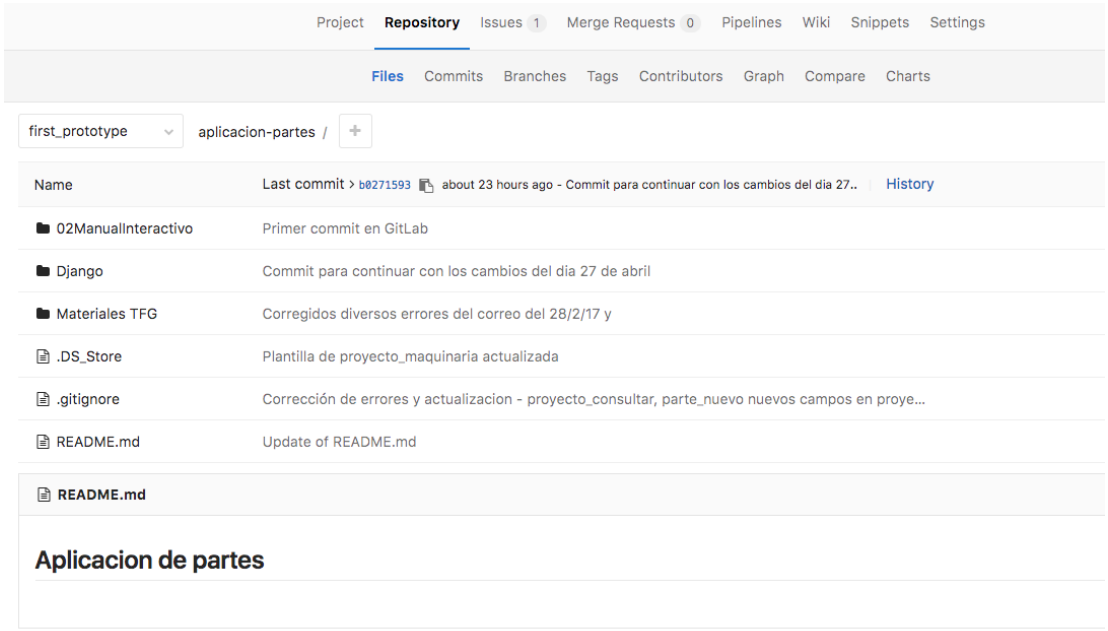


Figura 2.2: Resumen del proyecto GitLab

Desde el comienzo del TFG se han planificado reuniones semanales con el tutor y, aproximadamente, trimestrales con la empresa donde se iban analizando cada una de las etapas del desarrollo en espiral.

2.5. Plan de trabajo

El plan de desarrollo del proyecto ha pasado por las siguientes etapas:

- **Toma de contacto con las tecnologías de servidor:** Aterrizaje en la plataforma `Django` como framework de desarrollo y con `MySQL` como base de datos elaborando un pequeño servidor de pruebas que sirve páginas sencillas y guarda unos pocos campos.
- **Primer prototipo:** Desarrollo de un *primer prototipo* basado en información facilitada por la empresa. Con el prototipo cerrado, se concertó una reunión con la empresa para cerrar aquellas líneas donde no se tenía un objetivo claro y establecer nuevas metas en función de sus necesidades. Se crea un modelo de datos tentativo y un servidor básico.
- **Ajuste del modelo de base de datos:** Las interacciones con la empresa y con el pequeño grupo de pilotos que hacen uso de la aplicación, lleva a una reestructuración del primer modelo de la base de datos hasta alcanzar una configuración que se ajuste a todos los requerimientos. Con el modelo de base de datos estructurado de nuevo se ajusta el contenido de la aplicación para marcar un *segundo prototipo*.
- **Despliegue en el servidor en producción:** Con un servidor que sirve las páginas y almacena los datos de manera correcta y fiel al objetivo se procede a darle más robustez al mismo mediante los elementos de seguridad que aporta el servidor en producción de `Apache` así como sistemas de backup y monitorización, dando lugar al *tercer prototipo*.

- **Periodo de pruebas por parte de la empresa, en producción:** Finalmente y gracias a los pequeños reportes que van apareciendo mediante los test diarios y los que realiza la empresa por medio de los operarios que la usan, se liman flecos pequeños. El ajuste de estos reportes cierran la *solución final* de la aplicación web.

Capítulo 3

Infraestructura

En este capítulo, se explica la tecnología que se ha empleado en el desarrollo de la aplicación web con breves introducciones a cada una de ellas. Se ha optado por desarrollar la aplicación usando un entorno de desarrollo, metiendolo dentro de un servidor web en producción que le otorga robustez y seguridad.

3.1. Entorno Django

`Django`[1] es un entorno de desarrollo web de código abierto, escrito en `Python`, que respeta el patrón de diseño conocido como Modelo–vista–controlador. Fue desarrollado en origen para gestionar varias páginas orientadas a noticias de la World Company de Lawrence, Kansas, y fue liberada al público bajo una licencia BSD en julio de 2005. El entorno de desarrollo fue nombrado en alusión al guitarrista de jazz gitano Django Reinhardt. En junio de 2008 se anunció que la recién formada `Django Software Foundation` se haría cargo de `Django` en el futuro.

`Django` pone énfasis en el re-uso de código, la conectividad y extensibilidad de componentes, el desarrollo rápido. Un esquema simple del funcionamiento de un servidor `Django` sería como el que se ve en la figura 3.1. Puede verse como una solicitud del cliente llega al controlador del entorno, consulta a un modelo, que es un reflejo de lo que se aloja en la base de datos, y renderiza una plantilla `HTML` a través de la lógica programada en la vista. Como resultado devuelve, en forma de respuesta a los navegadores, el resultado de la petición. Para este proyecto se ha empleado la versión 1.9 del *software*. Cada uno de los elementos de este patrón de diseño se describirán en los siguientes puntos.
Dentro del entorno `Django` la lógica de funcionamiento del sitio web se programa en lenguaje `Python`, un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código legible. Se trata de un lenguaje de programación que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, usa tipado dinámico y es multiplataforma. En este proyecto se ha usado la versión 2.7 del lenguaje.

3.1.1. Modelo

El modelo es la única fuente de información de los datos que serán almacenados. Contiene los campos esenciales y los comportamientos de los datos que se guardarán. Generalmente, cada modelo se asigna a una tabla de base de datos única. Todo esto permite controlar el

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

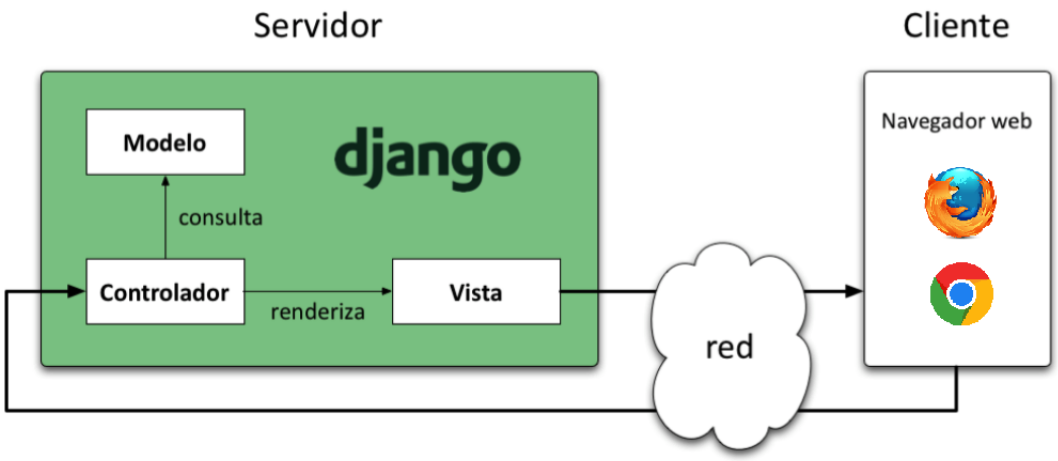


Figura 3.1: Esquema general del entorno Django

comportamiento de los datos. Puede verse un ejemplo simple de un modelo en el fragmento 3.1.

```
from django.db import models
class my_model (models.Model):
        field1 = models.CharField(max_length=200)
        field2 = models.BooleanField(default=False)
        field3 = models.FileField(upload_to="static/img")
        field4 = models.IntegerField(null=True)
        field5 = models.ForeignKey(Musician)
        ...
```

Fragmento 3.1: Ejemplo de modelo

Cada campo guarda un tipo concreto de datos necesario para su correcto almacenamiento en la base de datos que se quiera utilizar. Existen varios tipos: cadena de caracteres, booleano, archivo, número entero, etc o incluso direccionar a otro modelo (u otra tabla dentro de la base de datos) como *clave foránea*. Esto es importante para bases de datos relacionales (como la que se utiliza en esta aplicación) ya que no todos los datos se encuentran en una misma tabla y será necesario obtener información del resto.

Un cambio en el modelo de datos requiere una actualización de la base de datos, ya sea por la creación de un nuevo campo, alteración del nombre de una columna, etc. Para ello se requiere del comando `makemigrations` para detectar los cambios y `migrate` para efectuarlos. Ambos comandos son lanzados mediante el gestor `manage.py` que contiene una lista de funcionalidades, como puede ser crear la aplicación o lanzar el servidor, entre otras.

Una vez tenemos definido un modelo de datos es necesario decir a `Django` que va a ser usado. Esto se efectua en el archivo de configuración `settings.py` en la sección de 'aplicaciones instaladas' o `INSTALLED_APPS` como puede verse en el fragmento de código 3.2.

```
INSTALLED_APPS = [
    #...
    'myapp',
```

```
    #...
]
```

Fragmento 3.2: Configuración de la variable 'Aplicaciones Instaladas'

Con un modelo definido y configurado, la aplicación queda lista para poder almacenar información.

3.1.2. Controlador

El controlador de `Django` es el elemento que gestiona en primera instancia la petición `HTTP` ya que, en función de la página solicitada, encaminará esa petición a una u otra vista. La manera en la que selecciona una vista está relacionada con los elementos que reciba por la `URL`, la cual puede contener multitud de parámetros. Estas `URLs` pasarán por unas comprobaciones a través de lo que se denomina expresiones regulares (o `regex`), que son una secuencia de caracteres que forma un patrón de búsqueda.

```
url(r'^$', views.index, name=pagina_principal),
```

Fragmento 3.3: Ejemplo de expresión regular en urls.py

Para entender mejor este concepto, puede verse un ejemplo en el fragmento 3.3. En él puede verse cómo, si la página solicitada por el cliente cumple con la expresión regular `vacía`, la petición `HTTP` será encaminada a la vista de la página principal, `index`. Esto ocurre porque el patrón de búsqueda para este ejemplo está encerrado entre dos símbolos que hacen referencia a "`empieza por:` "(cuyo símbolo es `^`) y "`acaba por:` "(símbolo $).

De esta manera se hace un primer corte en el archivo del controlador (`urls.py`) por si la petición a una página determinada no tiene representación en la lista de URLs del controlador, elevando una excepción de página no existente.

```
url(r'^articulos/(?P<year>[0-9]{4})/(?P<month>[0-9]{2})/$', views.archivo_mensual),
```

Fragmento 3.4: Ejemplo más estricto de una URL

Estas expresiones pueden aumentar su complejidad mediante los parámetros que se pasen por la `URL` como puede verse en el ejemplo del fragmento 3.4.Para este caso una petición que empiece por `articulos/` tendrá que ir seguida de un valor anual válido de 4 cifras ({4}), seguido de un mes válido de 2 cifras. Esto conducirá a la vista `archivo_mensual`.

3.1.3. Vistas

Una función vista, o vista para abreviar, simplemente es una función de `Python` que toma una solicitud Web y devuelve una respuesta Web. Esta respuesta puede ser el contenido `HTML` de una página Web, un redireccionamiento, un error 404, un documento `XML`, una imagen, etc. La propia vista contiene la lógica arbitraria que sea necesaria para devolver esa respuesta. Este código puede vivir en cualquier directorio, siempre y cuando esté en una ruta de `Python` y dentro de una función. La convención es poner vistas en un archivo llamado `views.py`, colocado en el directorio del proyecto o de la aplicación.

```
def index(request):
    """ Pagina principal de la aplicacion """

    return render(request, 'templates/index.html')
```

Fragmento 3.5: Ejemplo de una vista simple

En el archivo `views.py` existen tantas vistas como URLs haya en el archivo `urls.py`. Puede verse un ejemplo de vista en el fragmento 3.5. Esta vista devuelve simplemente una página `HTML` ubicada en el directorio '`templates/`' llamada '`index.html`'. Más adelante se verán casos donde aumenta la lógica y complejidad de una función vista.

3.1.4. Plantillas

Un entorno web como Django necesita una manera conveniente de generar `HTML` dinámicamente. El enfoque más común es el de las plantillas. Una plantilla contiene las partes estáticas en formato `HTML` otorgando de esqueletización a la página, así como alguna sintaxis especial que describe cómo se insertará el contenido dinámico que aporta `Django`. Esta sintaxis consiste en el encerrado entre caracteres especiales de llaves seguido del símbolo %; {% ... %}. Estos elementos son los que dotan a la página web del dinamismo que requiere la aplicación. Esta sintaxis permite construir unas partes u otras del documento `HTML` en función de las peticiones del cliente. De esta manera, podemos tener una misma página para distintos tipos de usuarios cambiando la forma y el contenido de los datos que se entregan al navegador web, únicamente con el uso de estas variables de plantilla.

La manera que tiene `Django` de rellenar estas plantillas dentro de estas variables es mediante el `contexto` creado en la vista correspondiente. En él se almacenan todo tipo de variables, enteros, caracteres, arrays, búsquedas en la base de datos, etc, en un diccionario mediante pares: clave/valor. Este contexto se añade a la renderización y se completa la plantilla, retornando al cliente el documento `HTML` construido. La vista de la página principal quedaría como se ve en el fragmento 3.6.

```
def index(request):
    """ Pagina principal de la aplicacion """

        context = {
                        "autenticado": "true",
            "usuario" : "user",
            "registrado": "true",
            . . .
        }
    return render(request, 'templates/index.html', context)
```

Fragmento 3.6: Ejemplo de una vista con más complejidad

En el proceso de *renderización* se construye la página de inicio de la aplicación si el usuario está autenticado o la página de inicio de sesión si no lo está, como puede verse en el fragmento 3.7.

```
[ . . . ]
```

```
{% if autenticado == "true" %}
        . . .
        Contenido de la pagina web
        . . .
{% else %}
        <!--Formulario de inicio de sesion:-->
        <form method="POST" action="/login">
                <p>Usuario:</p>
                <input type="text" name="name">
                <p>Contrasena:</p>
                <input type="password" name="password">
                <button type="submit" Iniciar Sesion </button>
        </form>
{% endif %}

[ . . . ]
```

Fragmento 3.7: Ejemplo de variables de plantilla

De esta manera es como se contruye el contenido dinámico de la aplicación.

3.1.5. Localización e Internacionalización

Internacionalización (*i18n*) se refiere al proceso de diseño de programas para su uso desde distintas regiones en función del idioma. Esto incluye el marcado del texto (tales como elementos de la interfaz con el usuario o mensajes de error) para su futura traducción, la abstracción de la visualización de fechas y horarios de manera que sea posible respetar diferentes estándares. Localización (*l10n*) se refiere al proceso específico de traducir un programa automáticamente para su uso en una región particular en función del idioma configurado por el navegador.

`Django` en sí está totalmente internacionalizado. Todas las cadenas están marcadas para su traducción y existen variables de configuración que controlan la visualización de valores dependientes del idioma configurado como fechas y horarios. Dentro de las plantillas se pueden añadir cadenas de traducción personalizadas, representadas como {% *trans 'cadena_a_traducir'* %}. `Django` se encarga de usar esas cadenas marcadas para traducir los textos de las aplicaciones Web en tiempo de carga de acuerdo a las preferencias de idioma del usuario.

Dado que esta aplicación Web tendrá acceso desde fuera de España, el uso de esta herramienta que proporciona `Django` es de gran utilidad para que la página contenga textos dinámicos y configurables tanto por la aplicación como por necesidad del usuario.

3.2. MySQL

La base de datos empleada en este proyecto es MySQL en su versión 14.14, por su buena interacción con `Django`. `MySQL` es un sistema de gestión de bases de datos relacional desarrollado bajo licencia dual GPL/Licencia comercial por `Oracle Corporation` y está considerada como la base datos de código abierto más popular del mundo y una de las más populares en general junto a `Oracle` y `Microsoft SQL Server`, sobre todo para entornos de desarrollo web.

La ventaja del uso de `Django` con las bases de datos compatibles es el empleo del denominado: *Mapeo objeto-relacional* `ORM` (Object-Relational mapping). Esta técnica permite escribir código `Python` en lugar de SQL para hacer las consultas que necesita

la vista. Mediante los métodos `create()`, `save()`, `delete()` (entre otros) del objeto en cuestión invocados desde `Python`, se procede a modificar la base de datos.

Figura 3.2: MySQL como base de datos

A su vez, la simple gestión de copias de seguridad de `MySQL`, la cual se exporta con una línea de comandos `mysqldump`, es otro de los puntos a favor que presenta esta base de datos. En este TFG, la base de datos `MySQL` se usa para almacenar los partes de obra generados, que se pueden rellenar o consultar utilizando la aplicación web.

3.3. Apache

El servidor `HTTP Apache`, en su versión 2.4.18, es un servidor web `HTTP` de código abierto, para plataformas `Unix` (BSD, GNU/Linux, etc.), `Microsoft Windows`, `Macintosh` y otras, que implementa el protocolo `HTTP/1.12` y la noción de sitio virtual. Es un servidor multiplataforma, gratuito, muy robusto y que destaca por su seguridad y alto rendimiento.

La función de este servidor de alojamiento web es aceptar las peticiones de páginas (o recursos en general) que provienen de los visitantes que acceden al sitio web y gestionar su entrega o denegación, de acuerdo a las políticas de seguridad establecidas hacia la aplicación interior. Esto, que puede parecer simple, implica muchas facetas y funcionalidades que debe cubrir, como pueden ser:

- Atender de manera eficiente, ya que puede recibir un gran número de peticiones `HTTP`, incluyendo una ejecución multitarea ya que pueden darse peticiones simultáneas. Cualquier petición compleja (por ejemplo con acceso a base de datos) dejaría colapsado el servicio.
- Restricciones de acceso a los ficheros que no se quieran 'exponer', gestión de autentificaciones de usuarios o filtrado de peticiones según el origen de éstas.
- Manejar los errores por páginas no encontradas, informando al visitante y/o redirigiendo a páginas predeterminadas.
- Gestión de la información a transmitir en función de su formato e informar adecuadamente al navegador que está solicitando dicho recurso.
- Gestión de logs, es decir almacenar las peticiones recibidas, errores que se han producido y en general toda aquella información que puede ser registrada y analizada posteriormente para obtener las estadísticas de acceso al sitio web.
- Además, `Apache` permite configurar un *Hosting Virtual* basado en IPs o en nombres, es decir, tener varios sitios web en un mismo equipo (por ejemplo: nombreweb1.com, nombreweb2.com,....) o establecer distintos niveles de control de acceso a la información incluyendo el soporte a cifrado SSL utilizando protocolo seguro HTTPS.

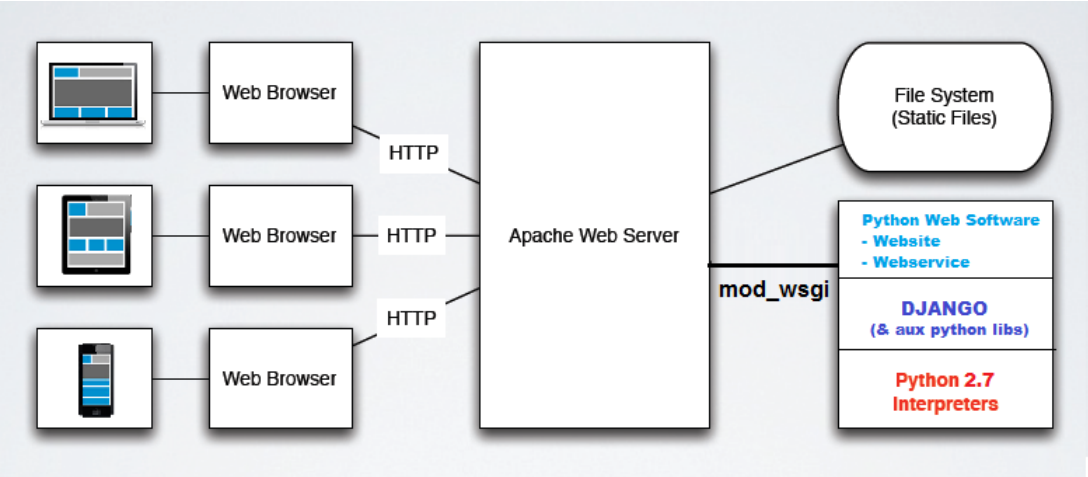


Figura 3.3: Diagrama de estructura de Apache

Puede verse un diagrama que ejemplifica la estructura general de una aplicación web en la figura 3.3.

Para este servidor se han utilizado diferentes módulos que completan la configuración y que se complementan muy bien con la aplicación `Django`. A continuación se muestran cada uno de ellos.

3.3.1. CGI (Common Gateway Interface)

`CGI` o `Interfaz de entrada común` es una importante tecnología de la World Wide Web que permite a un cliente (navegador web) solicitar datos de un programa ejecutado en un servidor web. Este módulo es necesario para que `Django` (o `Python`) pueda ejecutar órdenes del sistema operativo con el módulo '`os`' de `Python` ya que `Apache`, por defecto, no lo permite. De esta manera se hace una selección de aquellos ficheros que pueden ejecutar contenido en la máquina donde se encuentra alojado el servidor `Apache`. Esto aporta mayor control sobre qué elementos son ejecutados.

En el diagrama de la figura 3.4 puede verse cómo el servidor llama al interface con la orden (2) y devuelve la respuesta de un programa (3).

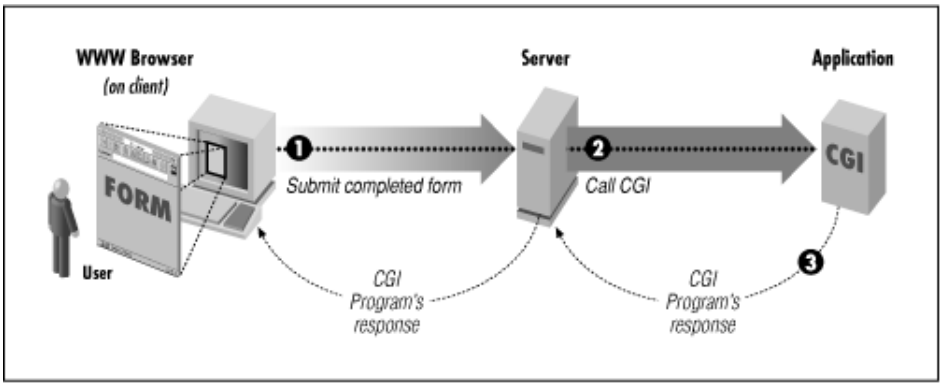


Figura 3.4: Diagrama de CGI

3.3.2. TLS (Transport Layer Security)

Transport Layer Security (TLS; en español «seguridad de la capa de transporte») y su antecesor Secure Sockets Layer (SSL; en español «capa de puertos seguros») son protocolos criptográficos que proporcionan comunicaciones seguras por una red. Esto permite que el navegador establezca comunicaciones seguras usando el protocolo de comunicaciones HTTPS.

Dado que en la aplicación existen comunicaciones con información privada, es necesario asegurar cada una de las interacciones con el servidor, por lo que se ha incluido esta capa de transporte que aporta cifrado.

Mediante el módulo de `Apache` '`mod_ssl`' se enrutan todas las peticiones al puerto por defecto de internet hacia el puerto donde serán tratadas las peticiones atendiendo a los protocolos de seguridad, por defecto el 443.

En el diagrama de la figura 3.5 se muestra una representación básica de los pasos que se siguen para establecer una comunicación segura con el servidor web en cada petición antes de pedir el recurso para que este lo devuelva de manera encriptada y segura para el cliente.

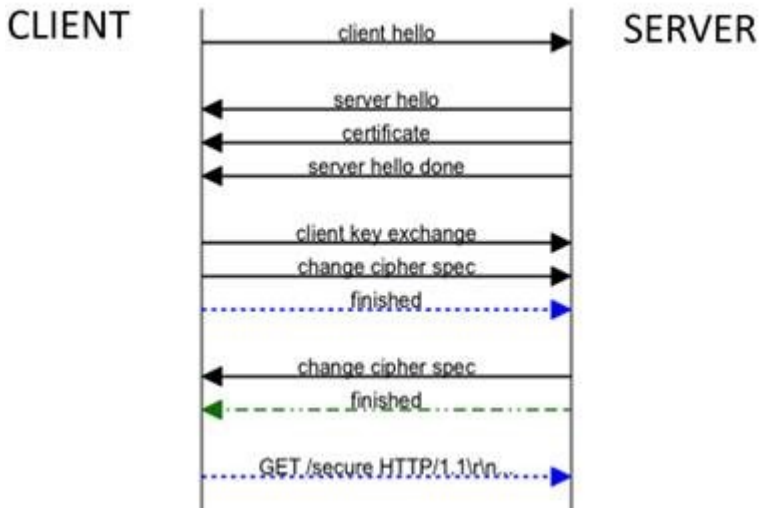


Figura 3.5: Modelo Handshake de comprobación de seguridad TLS

3.3.3. Despliegue de la aplicación. Módulo WSGI.

`WSGI` (Web Server Gateway Interface) es un módulo de `Apache` que provee una interfaz para correr aplicaciones Web en `Python` sobre `Apache`. Fue originalmente especificado en el PEP 333 por Phillip J. Eby y publicado en Diciembre de 2003. Es desde entonces cuando ha sido adoptado como un estándar para el desarrollo de aplicaciones web `Python` y usado en el entorno en el que se ha desarrollado este proyecto con `Django`.

De igual manera que ocurre con las bases de datos compatibles con el entorno `Django`, con el módulo `WSGI` es posible lanzar el servidor web `Django` de manera simple dentro de `Apache`. Únicamente hace falta configurar el archivo de `Apache` del dominio e indicar dónde se encuentra el archivo `wsgi.py` del proyecto. Con esto `Apache` lo lanzará como un

servicio en la máquina, en este caso, `Linux`.

Ese fichero de configuración puede verse en el capítulo 4 sección 4.4.1.

3.4. Interacción entre servidor y cliente web.

En este apartado se presentan dos tecnologías web usadas para la interacción entre cliente y servidor de la aplicación web desarrollada para distintos objetivos: `AJAX` u `WebSockets`.

3.4.1. AJAX

`AJAX`, acrónimo de *Asynchronous JavaScript And XML* (JavaScript asíncrono y XML), es una técnica de desarrollo web para crear aplicaciones interactivas o RIA (Rich Internet Applications). Estas aplicaciones se ejecutan en el cliente y mantienen comunicación asíncrona con el servidor en segundo plano. De esta forma es posible realizar cambios sobre las páginas sin necesidad de recargarlas completamente, mejorando la interactividad, velocidad y usabilidad en las aplicaciones.

En la figura 3.6 se muestra por un lado cómo existe una comunicación clásica entre cliente y servidor siguiendo el eje horizontal de tiempos. Estas peticiones tienen un tiempo de solicitud y respuesta donde no puede haber consultas entre medias. Por otro lado, en la parte inferior, además de la petición estandar `HTTP`, existen otras peticiones asíncronas con las iniciales.

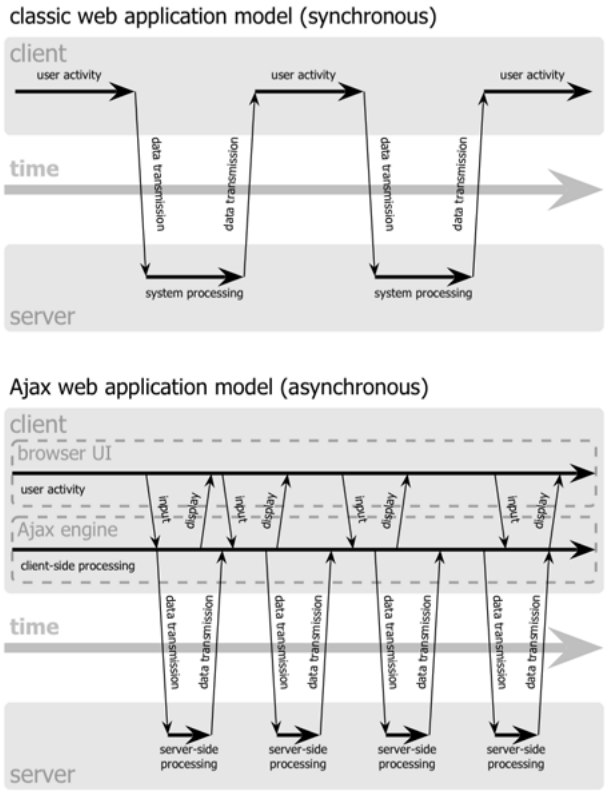


Figura 3.6: Diagrama de AJAX

Para el caso de este TFG, se emplea `AJAX` como interacción con el servidor, que consulta la base de datos de algunos de los campos en los formularios informando al usuario si el campo relleno es o no válido antes de que se produzca el envío.

3.4.2. WebSockets

WebSockets es una tecnología que proporciona un canal de comunicación bidireccional y *full-duplex* sobre un único *socket* TCP. Está diseñada para ser implementada en navegadores y servidores web, pero puede utilizarse por cualquier aplicación cliente/servidor. Debido a que las conexiones TCP comunes sobre puertos diferentes al 80 son habitualmente bloqueadas por los administradores de redes, el uso de esta tecnología proporciona una solución a este tipo de limitaciones proveyendo una funcionalidad similar a la apertura de varias conexiones en distintos puertos. Puede verse en la figura 3.7 un esquema del estableciemineto de conexión.

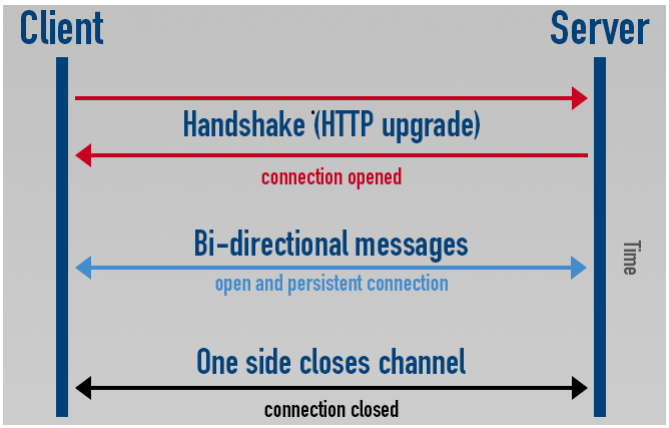


Figura 3.7: Diagrama WebSockets

En primer lugar el protocolo hace un estrechamiento de manos para abrir la conexión para dejar paso, a continuación, de la comunicación bi-direccional dejando una conexión persistente hasta que uno de los dos lados de la comunicación cierre la sesión. La especificación del protocolo WebSocket define dos nuevos esquemas de `URI`, `ws:` y `wss:` para conexiones no cifradas y cifradas.

Para este TFG, se han empleado `WebSockets` para gestionar por un puerto paralelo al del servidor el uso de notificaciones para el cliente a iniciativa der ser. Periódicamente el servidor consulta con una base de datos local determinadas condiciones. Si alguna de las condiciónes se cumple, el servidor tomará la iniciativa y enviará al cliente una alerta que será reflejada por el usuario en forma de alerta o '*pop up*'.

Capítulo 4

Servidor Web

En este capítulo se explica en detalle el funcionamiento del servidor que materializa la aplicación web de partes de obra así como los elementos y herramientas que dispone en su despliegue final en producción atendiendo a los objetivos marcados en el punto 2.

4.1. Diseño

Antes de describir los detalles de la aplicación web en el lado del servidor es necesario tener una visión general de los componentes de la misma así como los elementos en los que se centra este TFG (*Backend*).

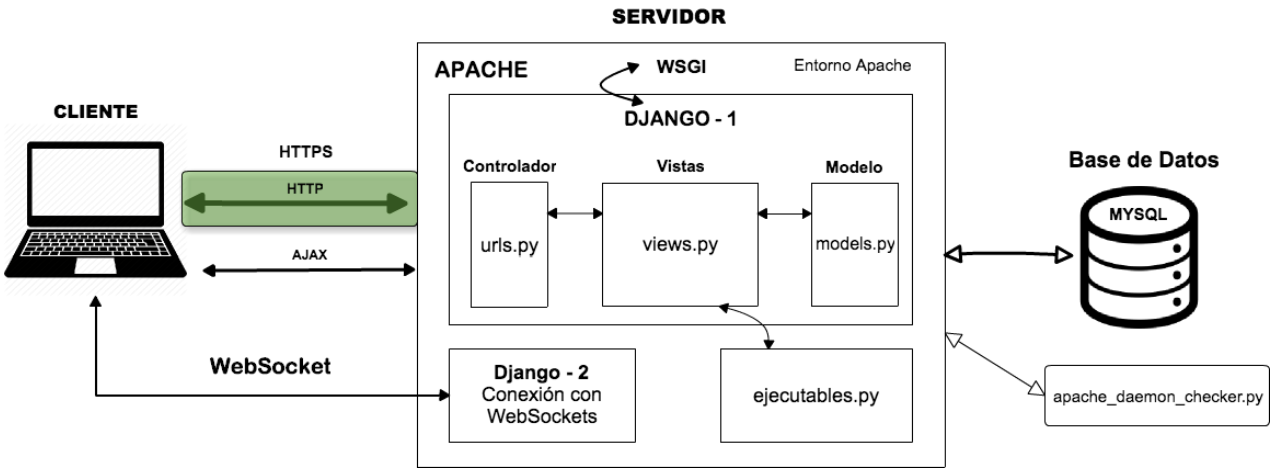


Figura 4.1: Arquitectura general de la aplicación web, haciéndo énfasis en el lado del servidor.

En la figura 4.1 se ven dos zonas claramente diferenciadas: el cliente, ejecutando en el ordenador del usuario, y el servidor, corriendo en la infraestructura de la empresa. Desde el lado cliente, un operario puede acceder a la aplicación desde cualquier dispositivo y lugar con conexión a internet que cuente con un navegador web con soporte `HTML5`.

El acceso al servidor se inicia mediante el protocolo HTTP con la capa de seguridad, pasando a ser HTTPS. Una vez en la aplicación, el servidor Django va devolviendo cada una de las páginas web que el cliente va solicitando, pudiendo consultar documentos, información de trabajadores, maquinaria o rellenar campos de formulario para ingresar nuevos registros. Alguno de los campos de estos formularios cuentan con la tecnología `AJAX` que efectúa peticiones asíncronas a la base de datos para que, antes de hacer efectivo un

formulario, comprobar si es válido ese campo, por ejemplo.

Por último, si el usuario tiene rol específico, podrá acceder a diferentes secciones de la aplicación.

Por otro lado el **Servidor** que es el entorno donde se centra este TFG. Si se mira desde fuera hacia dentro, en primer lugar tenemos el servidor en producción Apache. Es el encargado de gestionar las peticiones HTTPS que llegan desde el cliente. Este servidor hace de intermediario para gestionar las peticiones que vayan hacia el servidor Django-1, dentro suyo.

Dentro del servidor `Django` se hará cargo de la URL que se solicita el *controlador*. Este encaminará a una u otra *vista* la petición y se consultará a la *base de datos* respetando el *modelo* que la describe. La comunicación que realiza el servidor en producción `Apache` y el de gestión (`Django`) es mediante el módulo `WSGI` (Web Server Gateway Interface), que es un interfaz de comunicación entre el servidor web y la aplicación escrita en `Python`, en este caso `Django`.

En el mismo entorno de `Apache` se encuentra el fichero `ejecutables.py`. Este fichero ha sido externalizado de `Django` debido a que `Apache` no permite la ejecución de llamadas al sistema operativo a menos que se externalice y se permita usando el módulo CGI descrito en el 3.3.1. Ese fichero cuenta únicamente con dos funciones que realizan llamadas al sistema operativo.

A su izquierda, se cuenta con un segundo servidor `Django-2`, muy simple, encargado de las conexiones `WebSockets` mediante el módulo `Django Channels`. Estas peticiones siguen el protocolo '`ws`' que cuenta a su vez con su propia capa de seguridad, al igual que ocurre con el protocolo `HTTP` ('`wss`'). Es necesario un servidor de apoyo para estas peticiones, ya que `Apache` encuentra conflictos cuando le llegan peticiones de `WebSockets` con la capa de seguridad. El único propósito de este servidor de apoyo es el tratamiento de este tipo de solicitudes.

Por último, para asegurar la robustez del servidor en producción, está el fichero `fichero apache_daemon_checker.py` que periódicamente hace peticiones al sistema operativo comprobando el estado del servidor `Apache` e interviene en el caso de que no se encuentre activo entre los servicios, reiniciando, en ese caso, el servidor web.

4.2. Servidor Web Django - 1

En este servidor se encuentra la mayor parte lógica de la aplicación web, es el servidor principal. En él se encuentra el modelo de la base de datos, el controlador para administrar las peticiones de entrada y las vistas para devolver la información solicitada al cliente con las plantillas *renderizadas*. La generación de documentos Excel también se encuentra dentro de este servidor además de la orden para la exportación de los documentos en formato pdf. Este servidor Django - 1 se conecta con el servidor en producción mediante el módulo integrado en `Django`, `WSGI`.

4.2.1. Modelo, interacción con la base de datos.

Como se explicó en el 3, el modelo hace referencia a cómo se estructuran los datos en la base de datos. En la aplicación exiten 7 modelos que hacen referencia a 7 tablas en la base de datos `MySQL`. Existe una relación muy fuerte entre ellas. La relación y la dependencia que existe entre ellas puede verse en la figura 4.2. La dirección de las flechas indica de qué otra tabla depende su contenido para completarse. Por ejemplo, para la tabla 'Partes Obra', se necesita información de la tabla 'Maquinaria', 'Personal' e 'Intervalos'.

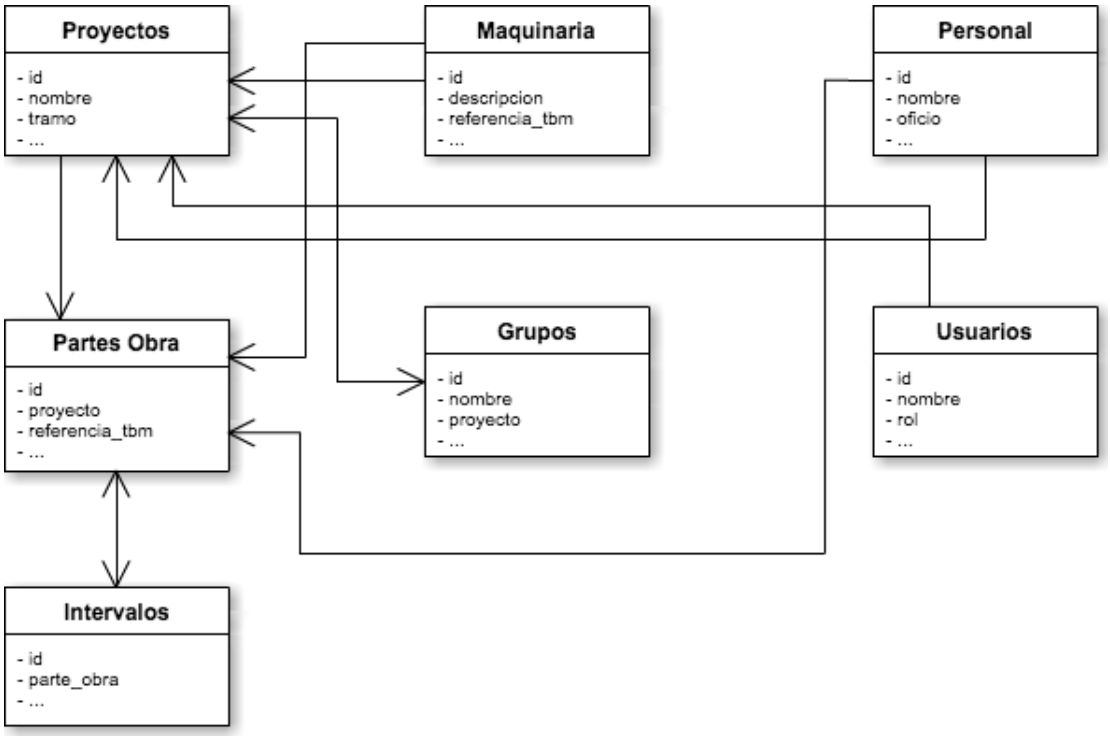


Figura 4.2: Diagrama de la Base de Datos

De todas ellas, la tabla en la cual se centra el objetivo principal tanto del TFG como de la aplicación es la de `'Partes de Obra'`. Esta tabla se alimenta del contenido del resto, además de generar el suyo propio. Está ligada al `proyecto` al que pertenece y es la base para poder generar los partes de obra de ese proyecto. El resto de tablas alimentan a la de 'Proyectos' y a partir de esa información, los partes de obra contendrán campos rellenos de modo automático simplemente por pertenecer a un proyecto en concreto.

Por otro lado, cada uno de los partes está compuesto por intervalos cerrados de horas donde ocurren *eventos* como pueden ser: `Excavación`, `Bajada de Tubo`, `Averías` o `Paradas` que describen el comportamiento de lo ocurrido en el turno mediante el relleno de campos. Esa dependencia entre la tabla de `Partes de Obra` y `Intervalos` es bidireccional ya que cada vez que se quiere exportar un parte de obra o consultarlo es necesaria esa tabla y, por otro lado, la tabla de intervalos no tiene sentido por si misma si no es con el parte asociado.

El fragmento 4.1 muestra un extracto del modelo de base de datos asociado a la tabla de 'Partes de Obra'.

```
class ParteObra(models.Model):
    nombre_proyecto = models.CharField(max_length=200, blank=True)
    tramo = models.CharField(max_length=200)
```

```
    piloto = models.ForeignKey(User, null=True, blank=True, related_name="jefe_obra")
    fecha = models.DateField(timezone.now, null=True)

    Dia = 'Dia'
    Noche = 'Noche'
    TURNOS = ( (Dia, 'Dia'), (Noche, 'Noche'),)
    turno = models.CharField(max_length=200, choices=TURNOS)

    hora_inicio_turno = models.CharField(max_length=200, null=True, default="8:00")
    hora_final_turno = models.CharField(max_length=200, null=True, default="20:00")

    tbm = models.ForeignKey(Maquinaria, null=True, blank=True, related_name="tbm")
    contenedor = models.CharField(max_length=200, null=True)

    # ================ VALORES FINALES ================================
    revest_inicial = models.FloatField(null=True)
    revest_final = models.FloatField(null=True)
    [ . . . ]


    # ========================= INTERVALOS ============================
    intervalos = models.CharField(max_length=10000, default=[])

    # ================== PERSONAL  ============================
    trabajadores_en_turno = models.CharField(max_length=1000)

    # ================== GEOLOGIA ============================
    Vacio = None
    Arcilla = 'Arcilla'
    Arenas = 'Arenas'
    . . .

    GEOLOGIA = (
        (Vacio, ''),
        (Arcilla, 'Arcilla'),
        (Arenas, 'Arenas'),
        [ . . . ]
    )

    geologia = models.CharField(max_length=200, choices=GEOLOGIA, null=True)

    # ================== CONSUMOS ============================
    consumo_gasoleo = models.FloatField(null=True)
    [ . . . ]

    # ================== NOTAS ============================
    trabajos_realizados = models.CharField(max_length=2000, null=True)
    [ . . . ]

    # ================== VALORES TOTALES ============================
    total_turno = models.FloatField(null=True)
    [ . . . ]
```

Fragmento 4.1: Modelo de Parte de Obra

Puede verse cómo los campos 'piloto' o 'tbm' dependen a su vez de las tablas que se veían en el fragmento 4.2 (Usuarios' y 'Maquinaria') mediante el tipo 'ForeignKey':

```
piloto = models.ForeignKey(User, null=True, blank=True, related_name="jefe_obra")
```

Fragmento 4.2: Ejemplo de búsqueda en otra tabla

Una de las ventajas de `Django` es su facilidad para interactuar con las bases de datos ya que abstrae al programador del lenguaje de consulta y acceso (SQL) y de las especificaciones de la base de datos concreta que se va a utilizar. Se usa el ORM de la librería de `Django` (en lenguaje `Python`) para interactuar de la misma manera independientemente de la base de datos. Es decir, para crear un objeto se emplea el mismo método: '`create()`' y, para guardar, '`save()`', ya se esté usando detrás una base de datos `MySQL` o una `SQLite` o una `MongoDB`, etc.

4.2.2. Controlador

A través de lo visto en el capítulo 3, apartado 3.1.2, el controlador de este servidor gestiona las vistas con el archivo `urls.py`. El fragmento 4.3 muestra un extracto creado para esta aplicación.

```
[ . . . ]
urlpatterns = [
        url(r'^$', views.index),
        url(r'^login', views.login),
        url(r'^logout', views.logout),
        url(r'^proyecto_abrir', views.proyecto_abrir),
        url(r'^proyecto_opciones/(\d+)', views.proyecto_opciones),
        url(r'^proyecto/consultar/(\d+)', views.proyecto_consultar),
        url(r'^proyecto/editar/(\d+)', views.proyecto_editar),
        url(r'^proyecto/trabajador_consultar/(\d+)/(\d+)', views.
            proyecto_trabajador_consultar),
        url(r'^proyecto/trabajadores/(\d+)', views.proyecto_trabajadores),
        url(r'^proyecto/maquinaria/(\d+)', views.proyecto_maquinaria),
        url(r'^proyecto/produccion/(\d+)', views.proyecto_produccion),
[ . . . ]
```

Fragmento 4.3: Ejemplo de URLs que pasan por el controlador

En la variable `'urlpatterns'` del fragmento 4.3 se encuentran todas las direcciones de la aplicación que, mediante las expresiones regulares (visto también en el capítulo 3), se enrutan hacia la vista en `Python` correspondiente. Este archivo es estructuralmente muy simple, pues únicamente contiene esa variable con cada una de las URLs de la aplicación con la vista asociada.

El archivo real completo contiene 62 enlaces que se corresponden con las 62 vistas de las que está compuesta la aplicación. Si un patrón de URL no contiene una vista asociada, el servidor levantará un error indicando que el enlace está incompleto por faltar su pareja.

Si tomamos como ejemplo el primer elemento, `views.index`, vemos que esa vista responde cuando la URL empieza (`^`) y termina (`$`) con la URL vacía, es decir, la raíz de la página web y, por tanto, la página principal.

En la novena línea del extracto 4.3, podemos ver cómo la condición se hace más estricta, teniendo que ir en la URL, además de la página a la que se quiere acceder dos parámetros de tipo número `(d+)` seguido uno detrás de otro y separado por '/'. Con esas condiciones se irá a la vista correspondiente que, para este caso es `views.proyecto_trabajador_consultar`.
Existen multitud de posibilidades que se pueden añadir para encontrar patrones en la expresión regular y hacer más exacta y estricta una condición en función de la información que se requiera.

4.2.3. Vistas, recepción de un parte de obra

Existen tantas vistas como URLs haya en el archivo `urls.py`, para el caso de la aplicación desarrollada, 62. Para mejor localización de cada una de ellas se ha optado por agruparlas por secciones; 'Proyectos' y 'Partes de Obra', 'Maquinaria', 'Personal', 'Administración' y las que tienen relación con contenido `AJAX`, para notificaciones , pruebas e internacionalización. Puede verse un sencillo esquema de las temáticas en la figura 4.3:

Por parte del usuario, desde su navegador web, rellena un formulario de `Parte de Obra` generando `Intervalos` que enviará al servidor. Una vez se reciba el formulario relleno

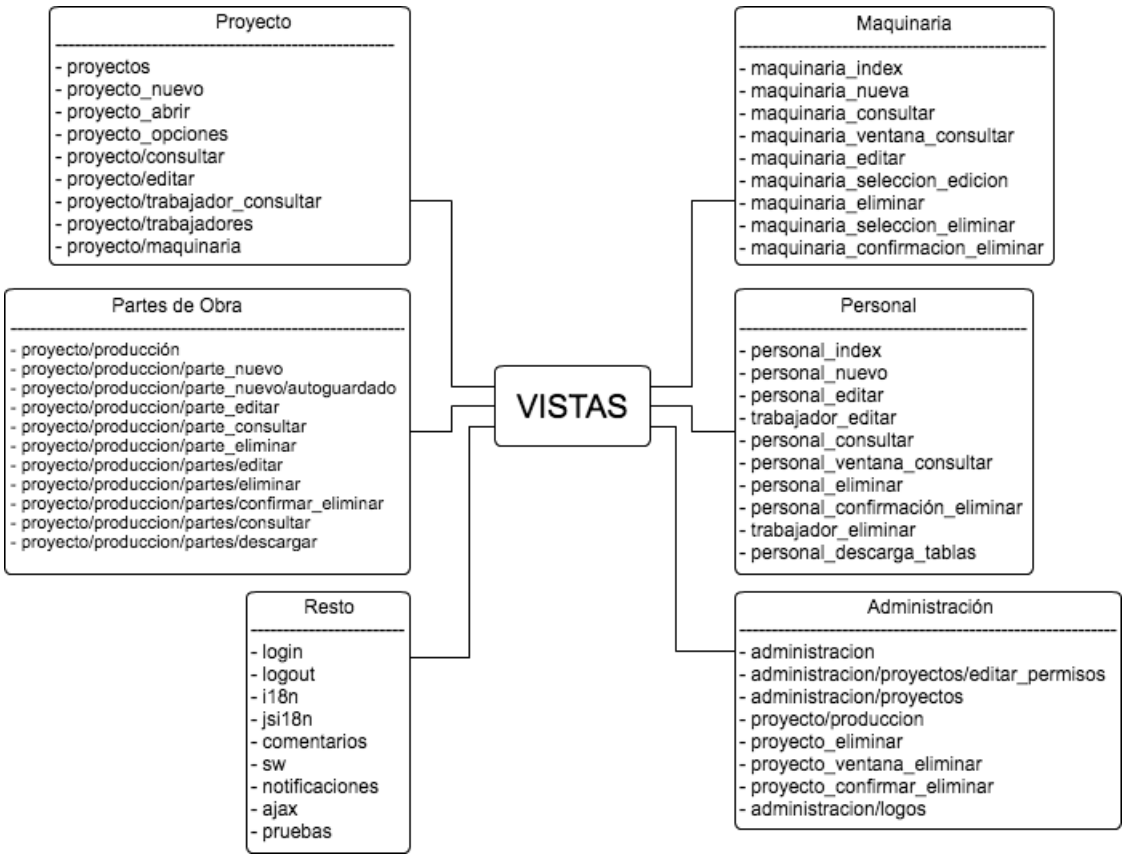


Figura 4.3: Diagrama de vistas enmarcado por temática

con el contenido, se verifican sus campos mediante sistemas de validación con el resto de tablas (comprobación de la existencia de `trabajadores`, `usuarios`, `maquinas`, etc) y se introducen una o varias entradas en la base de datos. Después, se generará el archivo `Excel` correspondiente al parte de obra introducido.

Siguiendo con el ejemplo de parte de obra, a continuación se irán desglosando las interacciones y detalles de las vistas involucradas en la creación de un parte de obra nuevo.

- **Paso 1 -** Se pasa por un filtro para saber si el usuario que está intentando acceder a la vista está autenticado y si además pertenece a alguno de los roles permitidos:

```
@user_passes_test(user_is_jefeobra_or_control_or_pilot)
```

Fragmento 4.4: Comprobación de usuario con rol específico

Una vez se concede el acceso a la vista, vemos que en la URL se están pasando tanto la solicitud (`request`) como un parámetro '`id`' que hace referencia al proyecto al que pertenece ese parte. Mediante el acceso a la base de datos se obtiene el objeto `proyecto` y, desde él se van obteniendo el resto de campo necesarios como el personal, maquinaria, etc, a través de los bucles que se ven en el siguiente fragmento de código:

```
def parte_nuevo(request, proyecto_id):
```

```
# Se obtiene el proyecto por el que nos pasan el ID
proyecto = Proyecto.objects.get(id=proyecto_id)

### MAQUINARIA - Tipo TBM ###
ids_maquinaria = json.loads(proyecto.maquinas)
maquinaria_proyecto = []

for id_maquina in ids_maquinaria:
    try:
        maquina = Maquinaria.objects.get(id=id_maquina)
        if maquina.tipo == "TBM":
            maquinaria_proyecto.append(maquina)
        else:
            pass
    except:
        ids_maquinaria.remove(id_maquina)

### TRABAJADORES ###
ids_trabajadores = json.loads(proyecto.trabajadores)
trabajadores = []

for id_trabajador in ids_trabajadores:
    try:
        trabajador = Empleado.objects.get(id=id_trabajador)
        trabajadores.append(trabajador.nombre)
    except:
        ids_trabajadores.remove(id_trabajador)
```

Fragmento 4.5: Búsqueda de Maquinaria y Trabajadores

Mediante el bucle for se recorre cada array y mediante el '`id`' se hace la búsqueda correspondiente a la tabla de '`Maquinaria`' y '`Trabajadores`', llenándose así los arrays de objeto '`Personal`' y objeto '`Maquinaria`'. Para asegurar que la búsqueda no dé error, se previene de los posibles fallos mediante `try/except` ya que puede ocurrir que, a la hora de obtener un trabajador, otro usuario lo haya eliminado de la base de datos elevándose un error y deteniendo la página. De esta manera, si eso ocurriera, lo eliminaría de la lista de ese proyecto y la aplicación continuaría su curso.

- **Paso 2 -** En función de en qué zona del 'parte nuevo' nos encontremos podemos estar ante la solicitud del formulario o ante la entrega del mismo. Para ello, en la vista se hace una clara diferencia mediante los métodos HTTP: `GET` y `POST`.

  - **Obtener el formulario (`GET`):** Esta primera opción es simple pues únicamente se busca en la base de datos a aquellos usuarios que están asociados a este proyecto en concreto y se devuelve el formulario con un contexto que ya contiene información que estará rellena cuando se cargue la plantilla. Ese contexto se alimenta de lo que se ha introducido en el fragmento anterior en las búsquedas en las tablas de '`Maquinaria`' y '`Trabajadores`'.

```
if request.method == "GET":

    ### USUARAIOS ###
    # Buscamos todos los usuarios
```

```
        all_users = User.objects.all()
        pilotos = []

        for user in all_users:
            if user.has_perm("prototype.is_pilot"):
                pilotos.append(user)

        context = {
            "authenticate" : "true",
            "proyecto" : proyecto,
            "trabajadores" : trabajadores,
            "pilotos" : pilotos,
            "maquinaria_proyecto" : maquinaria_proyecto,
        }

        return render(request, 'prototype/parte_obra_nuevo.html', context)
```

Fragmento 4.6: Opción GET. Se devuelve el formulario 'Parte de Obra'

- **Entregar el formulario(`POST`):** Para esta segunda opción, una vez es recibido el formulario relleno, se asigna el valor '0' a aquellos campos que vienen vacíos para que en la base de datos quede reflejado ese carácter (en el fragmento 4.7 (1)).

  A continuación, se comprueba que el formulario es válido comprobando que los campos que entran son del tipo que se especificó en el modelo previamente (en el fragmento 4.7 (2)). Además, en este punto se utiliza el método `cleaner_data` que limpia el formulario dejando únicamente los campos de `<input>` del `HTML` (los `'values'`).

  Posteriormente se calculan los valores de inicio y final del turno (en el fragmento 4.7 (3)) fijado por el usuario mediante los campos '`inicio_turno`' y '`final_turno`' para guardar la duración total para ese parte. Posteriormente, dado que el ficher `Excel` donde va a estar reflejado el parte contiene celdas para un turno de 12 horas, se fija a ese valor el turno. (Nota: Los partes físicos trabajan con una granularidad de 5 minutos por cada celda por lo que, para un turno de 12 horas, deja un valor total de 144 celdas).

  Por último, se crean e inicializan los arrays (fragmento 4.7 (4)) que alojarán los tramos del turno donde se haya producido trabajo. Dado que, actualmente, existen 4 tipos de intervalos, se crea uno para cada tipo.

```
# (1) Eliminamos los valores nulos del formulario y tramsformarlos por None
for key, value in request.POST.items():
    if value == "":
                value = 0
                request.POST.__setitem__(key, None)
        colocacion_datos_perforacion = []

    # (2) Formulario PARTE DE OBRA
    form = ParteObraForm(request.POST)
            if form.is_valid():
                form = form.cleaned_data;

                # (3) Se declaran los valores iniciales y finales del turno
                inicio_turno = request.POST.get("hora_inicio_turno")
                hora_inicio_turno = inicio_turno.split(":")[0]
                final_turno = request.POST.get("hora_final_turno")
                hora_final_turno = final_turno.split(":")[0]

                # Duracion del turno y el numero de celdas totales con ese
                    turno fijado
```

```
                duracion_turno = abs(int(hora_final_turno) - int(
                    hora_inicio_turno))

                # Fijamos el valor de las celdas a 144 (valor de 12 horas de
                    un turno)
                num_celdas_array = 144

                # (4) Representa el visor reconstruido a partir de las horas
                array_perforacion = []
                array_bajada_tubo = []
                array_averias = []
                array_paradas = []

                # Inicializacion de los arrays:
                for i in range(num_celdas_array):
                    array_perforacion.append('0')
                    array_bajada_tubo.append('0')
                    array_averias.append('0')
                    array_paradas.append('0')
```

Fragmento 4.7: Opción POST. Se obtienen campos del visor

- **Paso 3 -** Para cada intervalo se busca el tipo que entra (excavación, bajada de tubo, parada o avería) y se va adquiriendo cada uno de ellos por cada vuelta del bucle 'for' las horas de inicio y final. Con estos valores (no nulos) se llama a la función de *reconstrucción del visor* que se encarga de rellenar los arrays declarados en la sección anterior, reflejando cada uno de los intervalos introducidos. Esta función tomará los valores definidos del turno y de cada intervalo para posteriormente escribir en la hoja de cálculo.

  Después, se preparan los valores que generarán una escritura en la tabla '`Intervalos`' de la base de datos donde quedará reflejada una entrada por cada intervalo. Por último, en la tabla de '`Partes de Obra`' se almacenarán en un diccionario las referencias a cada uno de los intervalos introducidos mediante una *clave* (que será el 'ID' del intervalo que apunta a la tabla de 'Intervalos') y un *valor* (que será el tipo de intervalo almacenado).

  Este proceso se repite por cada tipo de intervalo; 'Avería', 'Bajada de Tubo' y 'Parada'. Puede verse lo explicado en el fragmento 4.8.

```
# Se recorre los intervalos de perforacion, bajada tubo, parada y averia. \
# Guarda el contenido en un diccionario con {id + tipo} y llama a la funcion de \
# reconstruccion del visor para que vaya actualizando

### INTERVALO PERFORACION ###
for i in range(num_intervalos):
    print "==========_INTERVALO_EXCAVACION_==========\n"
    try:
        tipo = "Perforacion"
        inicio_intervalo = request.POST.get("intervalo_" + str(i+1) + "_hora_inicio
            ")
        final_intervalo = request.POST.get("intervalo_" + str(i+1) + "_hora_final")
            if inicio_intervalo != None and final_intervalo != None:
            # FUNCION DE RECONSTRUCCION
                reconstruir_visor(array_perforacion, inicio_turno, inicio_intervalo
                    , final_intervalo, final_turno, tipo,
                    colocacion_datos_perforacion)

                objeto_intervalo = CrearIntervalo(request, "Perforacion", "
                    intervalo_" + str(i+1), form.get("fecha"))

                info_intervalo = {}
                info_intervalo["id"] = objeto_intervalo.id
                info_intervalo["tipo"] = "Perforacion"
                intervalos.append(info_intervalo)
    except:
```

```
        print "ERROR en intervalo de perforacion"

### INTERVALO AVERIA ###
    [ . . . ]

### INTERVALO BAJADA DE TUBO ###
    [ . . . ]

### INTERVALO PARADA ###
    [ . . . ]
```

Fragmento 4.8: Gestión de Intervalos

- **Paso 4 -** En el fragmento 4.9, se sigue sacando campos del formulario para buscar la información en la base de datos que permita completar el parte de obra. Se trata de los 'Pilotos' que deben ser usuarios existentes en la base de datos 'Usuarios'. Se realiza una búsqueda en la tabla de '`Empleados`' (donde no es necesario que sean usuarios dados de alta en el sistema, simplemente existir en la tabla 'Empleados' de la base de datos). Por último, el mismo procedimiento pero buscando en la base de datos de 'Maquinaria'.

```
### USUARIOS ###
if user_is_jefeobra_or_control(request.user):
    try:
        piloto = User.objects.get(first_name=form.get("piloto"))
    except:
        piloto = None
else:
    piloto = User.objects.get(first_name=request.user.first_name)

# Obtener los trabajadores del proyecto (el array)
trabajadores_turno = []
for nombre_trabajador in request.POST.getlist("trabajadores_en_turno"):
    trabajador = Empleado.objects.get(nombre=nombre_trabajador)
    trabajadores_turno.append(str(trabajador.id))

### MAQUINARIA ###
# Sacamos el valor de las TBM
tbms = request.POST.getlist("referencia_tbm")
tbm =  Maquinaria.objects.get(referencia_eh=tbms[0])
```

Fragmento 4.9: Búsqueda de usuarios y maquinaria en la base de datos

- **Paso 5 -** Con todos estos campos localizados se terminan de extraer los que no requieren una búsqueda ni comprobación en la base de datos (ya que simplemente son valores) y se procede a generar una entrada en la tabla '`ParteObra`' usando el `ORM` de `Django` con el método `create()`.

```
[ . . . ]
new_form = ParteObra.objects.create(
        # ========== INFORMACION BASICA ==========
    nombre_proyecto = form.get("nombre_proyecto"),
    tramo = form.get("tramo"),
        piloto = piloto,
        fecha = form.get("fecha"),
        turno = form.get("turno"),

[ . . . ]

)

generar_parte(new_form.id, array_perforacion, array_averias, array_bajada_tubo,
    array_paradas, colocacion_datos_perforacion)

        context = {
                "authenticate" : "true",
                "success" : "true",
                "mensaje" : "Parte de Obra enviado",
```

```
        }
        return redirect('/proyecto/produccion/' + proyecto_id)
```

Fragmento 4.10: Extracción de campos y llamada a Generar Parte

- **Paso 6 -** Como última funcionalidad dentro de esta vista, una vez guardado el parte de obra se llama a la función `generar_parte`. Esta función, usando la librería `openpyxl`, permite abrir una plantilla ya creada (y vacía) del parte de obra en formato `Excel` y escribir en aquellas casillas que se corresponden con los elementos registrados en la base de datos. De esta manera, queda un documento exportable en un formato editable (que posteriormente puede exportarse en formato `.pdf`) y que representa una unidad básica y completa de un turno de trabajo. Una vez construido el parte de obra en formato Excel se devuelve el contexto especificando al cliente que la entrada del parte ha resultado satisfactoria.

En este ejemplo se ha visto el procedimiento que sigue la vista cuando se accede y se guarda un Parte de Obra. El resto de vistas siguen una estructura lógica parecida de comprobación de elementos en las bases de datos y guardado de campos para crear un contexto y devolver una plantilla.

4.2.4. Plantillas

Cada petición realizada por el cliente y cada respuesta devuelta desde la vista por el servidor se hacen por medio de una presentación en formato `html` donde `Django` posee un lenguaje que permite representar la información que trae de vuelta y ayudar en su construcción. Son las `Plantillas`. Mediante este lenguaje de `Django` se pueden crear elementos `html` de manera dinámica mediante bucles o imponer condiciones de flujo mediante sentencias 'if'. Estas plantillas son interpretadas en el servidor gracias a unas etiquetas colocadas entre el lenguaje `html` y que se representa encerradas entre llaves y 'porcentajes' de esta manera: {% etiqueta %}.

Puede verse en el fragmento 4.11, un ejemplo de plantilla desarrollada donde se ejecutará un bucle `'for'` y dentro de él una condición de tipo `'if'`. Cada vuelta del bucle crea un elemento `html` (tipo tabla `<tr>`) que contendrá un campo del contexto que se construyó en la respuesta del servidor, estructurándose así, la tabla definitiva.

```
[ . . . ]

<table class="table table-hover">
  <thead>
      <tr>
        <th class="text-center">Fecha</th>
        <th class="text-center">Tramo</th>
        <th class="text-center">Turno</th>
        <th class="text-center">Acciones</th>
      </tr>
  </thead>

  <tbody id="lista_partes">
    {% for parte in partes_list %}
      <tr>
        <td>{{ parte.fecha }}</td>
        <td>{{ parte.tramo }}</td>
        <td>{{ parte.turno }}</td>
        <td><a href="/proyecto/produccion/partes/consultar/{{ proyecto.id }}/{{ parte.id
            }}">Abrir</a></td>
        <td><a href="/proyecto/produccion/partes/descargar/{{ parte.id }}">Descargar xls
            </a></td>
        <td><a href="/proyecto/produccion/partes/editar/{{ parte.id }}"></i>Editar</a></
            td>
```

```
        {% if perms.prototype.is_control or perms.prototype.is_jefeobra %}
        <td><a class="btn btn-danger" href="/proyecto/produccion/partes/eliminar/{{
            parte.id }}"></i>Eliminar</a></td>
        {% endif %}
      </tr>
    {% endfor %}
  </tbody>
</table>

[ . . . ]
```

Fragmento 4.11: Ejemplo de Plantilla con el etiquetado sin rellenar

Los elementos del contexto se ubica en la plantilla mediante la doble llave, {{'objeto.elemento'}}. El uso de los bucles y condiciones hace que por cada elemento del contexto se genere un nuevo elemento en la plantilla o se oculte información por no cumplirse cierta condición.

La tabla resultante de ejecutar el código anterior puede verse en cliente en la figura 4.4.

Fecha	Tramo	Turno	Acciones			
04/05/2017	Tramo1	Dia	Abrir	Descargar xls	Editar	Eliminar
06/04/2017	Tramo1	Dia	Abrir	Descargar xls	Editar	Eliminar
06/04/2017	Tramo1	Dia	Abrir	Descargar xls	Editar	Eliminar
06/04/2017	Tramo1	Dia	Abrir	Descargar xls	Editar	Eliminar
06/04/2017	Tramo1	Dia	Abrir	Descargar xls	Editar	Eliminar

Figura 4.4: Tabla resultante de ejecutar código de plantilla

4.2.5. Generación del fichero Excel

Una vez se construye el parte de obra, se envía al servidor y se obtienen los campos para obtener los valores que serán guardados en la base de datos. La obtención de esos datos y los cálculos realizados para generar los arrays de intervalos se utilizan en la misma vista para llamar a la función que generará el Excel correspondiente a ese parte. Esta función es `generar_parte`.

El uso de la librería `openpyxl` permite abrir, editar y guardar archivos Excel. Utilizando una plantilla predefinida, se carga el contenido en una variable, donde usando las coordenadas de las celdas de Excel se escriben los valores. Con el objeto '`parte`', que ha sido recibido de la vista, un fragmento que permite ver el funcionamiento de la librería sería el que se ve en el marco 4.12.

```
# INFORMACION BASICA
ws1['M1'] = parte.nombre_proyecto
ws1['CG1'] = parte.tbm.descripcion
```

```
ws1['DX1'] = parte.contenedor
ws1['M2'] = parte.tramo
```

Fragmento 4.12: Ejemplos sencillo de escritura en el documento Excel

Para representar los valores de los intervalos guardados en los array, la función pasa a ser un bucle '`while`' que se recorre tantas celdas como valores positivos tenga el array del intervalos. La variable de giro es una lista con cada una de las celdas que puede ser escrita. En función del tipo de intervalo que vaya a ser pintado, el bucle irá seleccionando aquellas celdas que cumplan la condición, dando lugar a la representación en Excel de igual manera que se tiene en la página web del parte de obra. Puede verse un fragmento del código en el marco 4.14.

```
[ . . . ]

i = 0
while i < len(letra_celda):
    if array_perforacion[i] == "1":
        num_celda = "8"
        ws1[letra_celda[i] + num_celda] = "X"
        try:
            if array_perforacion[i+1] == "0":
                num_slot = i
                hora_final_turno = num_slot/12
                finales_intervalo_perforacion.append(hora_final_turno
                    )
        except:
            finales_intervalo_perforacion.append(11)

    if array_bajada_tubo[i] == "2"
        num_celda = "9"
    [ . . . ]

    i = i + 1
```

Fragmento 4.13: Bucle para representar los valores de los intervalos

Para los campos de los trabajadores el proceso es muy similar. Mediante un bucle que se recorre el elemento del parte que contiene a todos los trabajadores, se van realizando peticiones a la base de datos para obtener el objeto de esa tabla para posteriormente pintarlo en la fila del documento Excel correspondiente como se ve en el fragmento 4.14.

```
# Se recorre el array de trabajadores del Parte de Obra y se solicita a la base
# de datos el objeto trabajador con el nombre de cada uno de los empleados:

for nombre_trabajador in trabajadores_en_turno:
        trabajador = Empleado.objects.filter(nombre=nombre_trabajador)

        # Se escoge el trabajador del turno que tiene el oficio de Jefe de Obra
    # y lo pintamos en la casilla correspondiente:

    if trabajador[0].oficio == "Jefe_de_Obra":
                if ws1['N54'].value == None:
                        ws1['N54'] = trabajador[0].nombre
        else:
```

```
            ws1['N53'] = "JEFES_DE_POZO"
            ws1['N54'] = ws1['N54'].value + "  " + trabajador[0].nombre

        # Se guarda en un array el resto de trabajadores que no tengan el rol
    # de Jefe de Obra y cuyo nombre no coincida con el del piloto, el cual,
    # ya esta pintado

    elif nombre_trabajador != parte.piloto.first_name:
                objects_trabajador.append(trabajador)
```

Fragmento 4.14: Bucle para representar los valores de los intervalos

El proceso se repite por cada uno de los elementos que tienen representación en el documento vacío buscando por un lado el elemento a ser introducido y por otro la coordenada correspondiente a la celda.

Como último paso, se procede a guardar el documento escrito con un nombre distinto al de la plantill, acomodándolo al proyecto en cuestión así como la fecha de creación como se ve en el fragmento 4.15:

```
archivo = "path/to/folder" + "parte_dia_" + nombre_proyecto + "_" + str(parte.fecha) + "_
    " + parte.turno + "_" + str(parte.id) + ".xlsx"
archivo = str(archivo)
wb_parte.save(archivo)
```

Fragmento 4.15: Guardado del documento Excel creado

El resultado final es un documento Excel como el de la figura 4.5 donde se puede ver el documento relleno con los campos que han sido introducidos por el usuario desde su navegador web, remotamente.

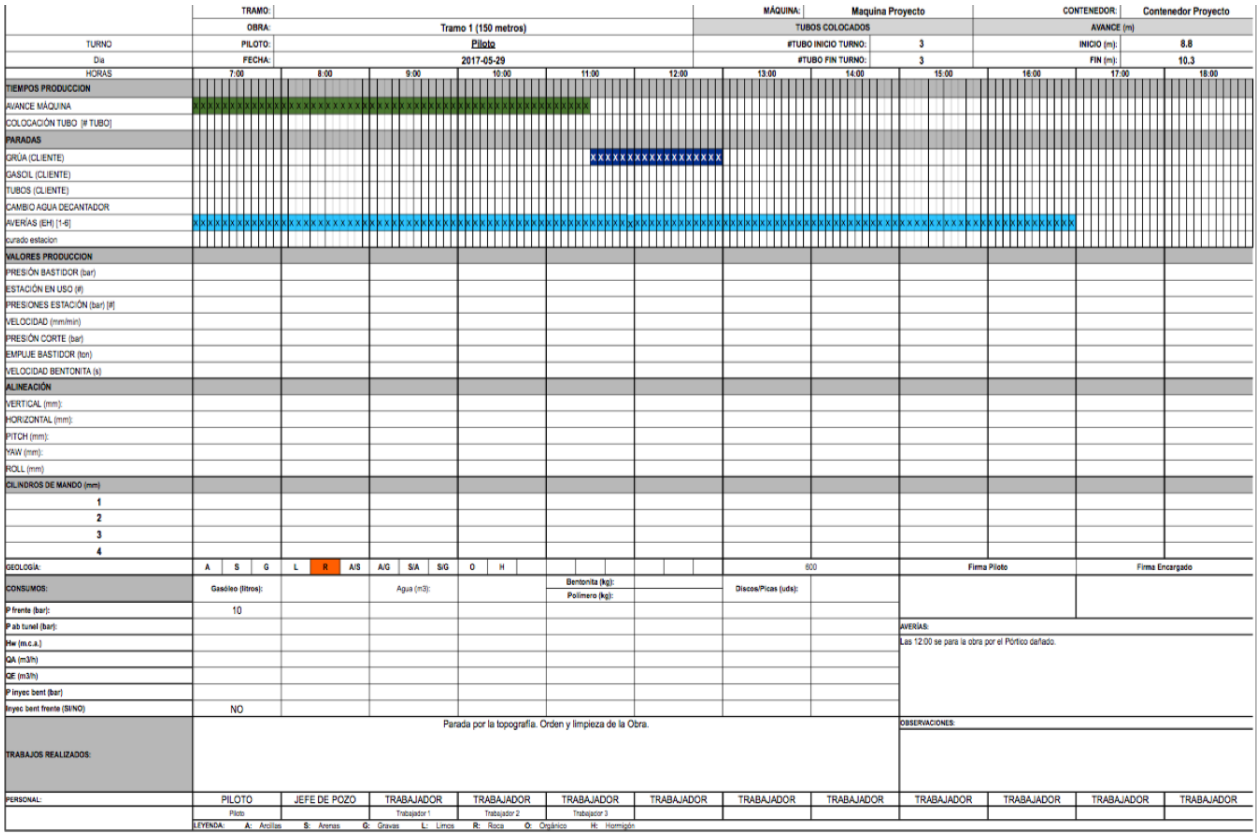


Figura 4.5: Parte de Obra relleno a su paso por la función 'generar parte'

4.2.6. Localización e Internacionalización

Dado que esta aplicación tiene acceso internacional y no todos los usuarios hablan Castellano, se ha añadido como funcionalidad la **Localización** e **Internacionalización** en el servidor `Django` dando soporte al idioma Inglés, y extensible a otros idiomas mediante las librerías `l10n` para localización e `i18n` para internacionalización. La primera de ellas permite establecer el idioma automáticamente en función del idioma del navegador, localizando en qué lugar se encuentra el cliente y devolviendo la página con el idioma correspondiente. También permite cambiar el idioma de la aplicación en cualquier momento mediante un selector.

Para esta extensión de la aplicación no se requiere instalación de ningún tipo de paquete de Python, simplemente ajustar la configuración de `Django` para que entienda que la página puede tener más de un idioma activando los paquetes (en `Django` denominado 'aplicaciones') que ya vienen instalados cuando se adquiere el entorno al comenzar el proyecto.

En el archivo `settings.py` se tiene que comprobar los siguientes elementos:

- Añadir `'django.middleware.locale.LocaleMiddleware'` a la lista de `Middlewares` que se tiene en la variable `MIDDLEWARE_CLASSES` debiendose colocar después de `'SessionMiddleware'`, ya que LocaleMiddleware hace uso de los datos de sesión. Puede verse cómo quedaría la variable en el fragmento 4.16:

```
    [ . . . ]
    MIDDLEWARE_CLASSES = (
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',

    # Language middleware package
    'django.middleware.locale.LocaleMiddleware',

    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.auth.middleware.
        SessionAuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
    'django.middleware.security.SecurityMiddleware',
)
    [ . . . ]
```

Fragmento 4.16: Configuración de los MiddleWare para la internacionalización

- Especificar el conjunto de idiomas o traducciones que tendrá la aplicación ya que esa variable puede ser accesible desde el cliente, ofreciéndole una lista de los idiomas disponibles en el servidor.:

```
[ . . . ]
LANGUAGES = (
 ('es', _('Espanish')),
 ('en', _('English')),
 . . .
)
```

```
[ . . . ]
```

Fragmento 4.17: Lenguajes configurados en esta aplicación web

- Verificar que estas variables estén con el valor `True` para hacer uso de la internacionalización (vienen por defecto así configuradas):

```
[ . . . ]
USE_I18N = True

USE_L10N = True

USE_TZ = True
[ . . . ]
```

Fragmento 4.18: Estado de los protocolos de localización e internacionalización activos

- Añadir la ruta a las carpetas donde se guardarán los archivos con los textos de la aplicación que hayan sido marcados mediante la variable `LOCALE_PATHS` definiendo la ruta de la carpeta `"locale"`:

```
[ . . . ]
LOCALE_PATHS = (
  os.path.join(BASE_DIR, "locale"),
)}
[ . . . ]
```

Fragmento 4.19: Directorio donde se encuentran los archivos con los textos

- Y finalmente definir en `settings.py` el procesador de contexto del i18n:

```
[ . . . ]
from django.conf import global_settings
TEMPLATE_CONTEXT_PROCESSORS = global_settings.TEMPLATE_CONTEXT_PROCESSORS + ('
    django.core.context_processors.i18n',)
}
[ . . . ]
```

Fragmento 4.20: Configurar el procesador de textos para i18n

Como siguiente paso, el servidor tiene que entender que la página que se solicita requiere de un cambio de textos por lo que en el **controlador** (`urls.py`) se debe añadir la siguiente entrada a la lista de URLs:

```
[ . . . ]
urlpatterns = patterns('',
  url(r'^i18n/', include('django.conf.urls.i18n')),
)
[ . . . ]
```

Fragmento 4.21: URL para cambio de textos

Para saber los textos que necesitan de traducción se usa el marcado en las plantillas `html`. Al comienzo de cada plantilla se carga la librería '`l10n`' e '`i18n`' (mediante las etiquetas`{% load l10n %}`, `{% load i18n %}`). Posteriormente, se marca todo el texto del que se requiere una traducción con la etiqueta `{% trans "texto" %}` para las plantillas `html` y `gettext('texto')` para los archivos `js`.

Como último paso, se requiere pasarle al servidor de `Django` unos comandos para que se generen y 'compilen' los mensajes marcados. Para crear los mensajes marcados se usa el comando del fragmento 4.22:

```
django-admin makemessages --locale 'en' -d django
```

Fragmento 4.22: Orden para la construcción de los mensajes marcados

Donde la orden `makemessages` creará los mensajes en la carpeta que se especifique mediante el parámetro `--locale`, en este caso 'en' de 'Inglés' y el tipo de archivo (`html` o `JavaScript`) con `-d django` o `-d djangojs`. En la carpeta 'en' se encuentra un archivo `.po` con los textos marcados como se ve en el fragmento 4.23.

```
...

#: prototype/templates/prototype/maquinaria_index.html:20
#: prototype/templates/prototype/personal_index.html:21
#: prototype/templates/prototype/proyecto_maquinaria.html:33
#: prototype/templates/prototype/proyecto_nuevo.html:32
msgid "Pagina Principal"
msgstr "Main Page"

...
```

Fragmento 4.23: Ejemplo de los textos creados en las palabras marcadas

La línea `msgid` muestra el mensaje marcado y la línea `msgstr` el resultado de la traducción que se necesite. Como puede verse, encima de cada texto marcado aparece la ruta a todos los archivos que contienen la misma palabra o frase marcada y la línea donde se encuentran, reduciéndose el número de veces que se tiene que traducir esa línea si la misma cadena aparece repetida en más de una ocasión.

Por último y para finalizar el proceso ofreciendo ya la página traducida, queda 'compilar' los mensajes mediante la orden del fragmento 4.24.

```
django-admin compilemessages
```

Fragmento 4.24: Compilar los mensajes traducidos

Con esto ya se tiene la página con más de un idioma sin necesidad de haber instalado ningún software adicional.

4.3. Servidor Auxiliar Django - 2

El sistema de notificaciones de la aplicación web viene administrada por un servidor secundario y muy simple de apoyo, `Django-2`. Éste servidor contiene únicamente la gestión de las notificaciones, que se mandan al cliente periódicamente cuando se cumplan unas condiciones.

Cuando el cliente accede a la aplicación realiza dos tipos de peticiones, la del recurso que está pidiendo mediante el protocolo `HTTP` y al servidor secundario mediante el protocolo `WebSockets`. La segunda petición va dirigida al servidor `Django-2` y es contestada por el servidor si en la base de datos existen trabajadores con un campo del tipo '*fecha*' caducado. Esto se verá reflejado en el cliente instantáneamente mediante el encendido de

Figura 4.6: Mensaje de Alerta en el cliente cuando llega una notificación

una alerta en la parte superior derecha como se ve en la figura 4.6.

La librería encargada de la gestión de las notificaciones es `Django Channels` que permite establecer canales bidireccionales usando la tecnología `WebSockets`. Esta librería se instala a través del administrador de paquetes `pip` de `Python` y se configura en `Django` como una extensión, añadiéndola a la variable de aplicaciones installadas (`INSTALLED_APPS`) que se encuentra en el archivo `settings.py`.

El archivo de configuración de la librería es `consumers.py` que contiene los métodos que son usados en la comunicación. Estos métodos son: `connect`, `keepalive` y `disconnect` como puede verse en el fragmento 4.25.

```
def websocket_connect(message):
    ChannelGroup("notifications").add(message.reply_channel)

    if(BuscarCaducados()):
        mensaje = {"contenido": "Empleados con documentos proximos a
            caducar", "id":"1"}
        ChannelGroup("notifications").send({
            "text": json.dumps({
                "id": 1,
                "content": "Empleados con documentos proximos a
                    caducar"
            })
        })

# Connected to websocket.keepalive
def websocket_keepalive(message):
    ChannelGroup("notifications").add(message.reply_channel)

# Connected to websocket.disconnect
def websocket_disconnect(message):
    ChannelGroup("notifications").discard(message.reply_channel)
```

Fragmento 4.25: Métodos del archivo de configuración consumers.py

El proceso arranca con el demonio, ubicado en el fichero `tasks.py`. En él se encuentra la función `BuscarCaducados()` que es llamada una vez al día por el demonio y que comprueba si la diferencia entre la fecha actual y la guardada en el campo de cada trabajador es menor a 15 días. En caso afirmativo, se genera una lista con los trabajadores que cumplen esa condición y se envía al cliente en formato `JSON` mediante el protocolo de comunicación `WebSockets` como puede verse en el fragmento 4.26.

```
[ . . . ]
while (d.isAlive()):
```

```
    # Tiempo en el que el daemon duerme hasta la siguiente
        comprobacion.
    time.sleep(86400)
    if(BuscarCaducados()):
        mensaje = {"contenido": "Empleados con documentos proximos a
            caducar", "id":"1"}
        ChannelGroup("notifications").send({
            "text": json.dumps({
            "id": 1,
            "content": "Empleados con documentos proximos a caducar"
            })
        })
[ . . . ]
```

Fragmento 4.26: Bucle de comprobación del campo para su validación

Dado que este servidor sólo se ocupa de comprobar los estados de los campos de interés, el número de URLs y vistas se reducen a una única puesto que todas las operaciones se realizan con los archivos `consumers.py` y tasks.py.

4.4. Servidor en Producción con Apache

En este apartado se detalla la configuración y el funcionamiento del servidor en producción `Apache` así como los scripts que garantizan fiabilidad ante caídas, seguridad en las transacciones `HTTP` y backups en la base de datos.

El servidor de producción elegido para esta aplicación ha sido Apache por ser uno de los más usados por su sencillez en la configuración además de la funcionalidad que aporta y de su buena integración con `Django` (más concretamente con el módulo `WSGI`).

Como se vio en la figura del apartado 4.1, `Apache` conforma un entorno de trabajo donde se establecen los distintos elementos de los que está compuesto el servidor global garantizando un control de cada programa y script que está siendo ejecutado. `Apache` divide su funcionalidad y componentes en unidades independientes que pueden ser configuradas por separado. La unidad básica que describe un sitio individial o dominio se denomina `virtual host`. Estas asignaciones permiten al administrador utilizar un servidor para alojar varios dominios o sitios en una simple interface o IP utilizando un mecanismo de coincidencias. Esto es relevante para cualquiera que busque alojamiento para más de un sitio en una sola dirección IP.

Cada dominio configurado apuntará al visitante a una carpeta específica que contiene la información de ese sitio, nunca indicará que el mismo servidor es responsable de otros sitios. Este esquema es expandible sin limites de software, tanto como el servidor pueda soportar la carga.

4.4.1. Conexión con el servidor Django-1 - WSGI

Primeramente se le indica a `Apache` dónde se encuentra alojado el proyecto `Django` además de indicar la ruta al interface de comunicación `WSGI`, a través de varias rutas a los directorios como se ve en el fragmento de código 4.27:

```
<IfModule mod_ssl.c>
    <VirtualHost _default_:443>
        # Django Configuracion:
        # ======================
        Alias /static /path_to_static_folder/
        <Directory /route_to_static_folder>
            Allow from all
            Require all granted
        </Directory>

        <Directory /path_to_wsgi_module>
            <Files wsgi.py>
                Require all granted
            </Files>
        </Directory>

        WSGIDaemonProcess mysite python-path=/
           path_to_project_folder_root:pwd/
           path_to_virtual_environment/
        WSGIProcessGroup mysite
        WSGIScriptAlias / /path_to_wsgi.py

        . . .
</IfModule>
```

Fragmento 4.27: Módulo WSGI en el servidor Django-1

En primer lugar se establece la ruta a los *archivos estáticos* (como son imágenes, archivos JavaScript, librerías, etc) otorgando los permisos para poder acceder a este directorio. Seguidamente, y de misma manera, se indica la ruta al módulo que permite ejecutar `Django` dentro de `Apache`, el módulo `WSGI`. Por último, se configuran los parámetros de este módulo `WSGI` cargado en `Apache` mediante la *ruta al directorio raíz* del proyecto para que se quede activo como un demonio dentro del sistema operativo, estando continuamente desplegado.

4.4.2. Conexión con el servidor Django-2 - Módulo para WebSockets

Dentro del fichero de configuración de `Apache` se encuentra el módulo que realiza las redirecciones al servidor `Django` secundario (`Django - 2`) mediante el protocolo `WebSocket` encaminándola a la ruta donde se encuentra la vista que responde a la petición `WS`. El módulo encargado de esta redirección se denomina `mod_proxy` y se configura indicando la URL por la que se pregunta y encaminando a la que se desea. Si se espera que el servidor conteste, se encaminará la respuesta de la misma manera mediante el `ProxyPassReverse`. Puede verse la redirección en el fragmento 4.28.

```
[ . . . ]

# MODULO DE REDIRECCION DE WEBSOCKET (mod_proxy)
# ==============================================
<IfModule mod_proxy.c>
    ProxyPass "/notificaciones/" "ws://127.0.0.1:8001/notificaciones/
        "
```

```
    ProxyPassReverse "/notificaciones/" "ws://127.0.0.1:8001/
        notificaciones"
</IfModule>

[ . . . ]
```

Fragmento 4.28: Módulo de redirección para WebSockets

Dado que esta comprobación se realiza periódicamente, se establece una conexión persitente y bidireccional entre cliente y servidor, siendo este último el que toma la iniciativa cuando se cumple el criterio en la base de datos, como se vió en el apartado 4.3. Para establecer una conexión `WebSocket`, el cliente manda una petición de negociación `WebSocket`, y el servidor manda una respuesta de negociación `WebSocket`.

4.4.3. Módulo CGI - Generación de los partes de obra en pdf

Además de la creación de archivos en formato `Excel` cuando se almacenan los datos, la aplicación web también permite visualizar y descargar el Parte de Obra guardado en formato `pdf`. Con esta funcionalidad se tiene acceso al contenido del parte de obra de manera online para su visualización o descarga siendo una réplica del que se tiene en formato `Excel`.

Para obtener el archivo en formato `pdf`, se utiliza un módulo de `Python`, que permite llamadas al sistema operativo usando los módulos de `Python os` y `sys`. Como medida de seguridad, `Apache` no permite que un elemento de la aplicación pueda ejecutar órdenes del sistema operativo a menos que sea consentido por el desarrollador, dándole un permiso especial en el archivo de configuración. De esta manera, se tiene un único archivo con estos privilegios de ejecución. El comando que permite la visualización y descarga, hace uso del software `LibreOffice`, que permite exportar archivos en el formato `.pdf`. El contenido del archivo que realiza la exportación puede verse en el fragmento 4.29.

```
import os, sys

def llamadas_sistema(nombre_proyecto, fecha_parte, turno_parte, id_parte):
    [ . . . ]
    sentencia = "libreoffice --headless --convert-to pdf /path_to_static_folder/" + "
        parte_dia_" + nombre_proyecto + "_" + str(fecha_parte) + "_" + str(turno_parte) +
         "_" + str(id_parte) + ".xlsx --outdir /path_to_other_static_folder"
    os.system(sentencia)
```

Fragmento 4.29: Función de generación de pdf

La sentencia es simple pues únicamente crea una variable pasándole el comando a `LibreOffice` y los parámetros `--convert-to pdf`, la ruta al archivo y el nombre para, posteriormente, ejecutarlo con `os.system` y convertir un `.xlsx` (del formato `Excel`) en un `.pdf`.

El módulo encargado de permitir al servidor `Django-1` ejecutar la sentencia descrita en el fragmento 4.29 es `CGI` (Common Gateway Interface). Puede verse en el fragmento 4.30 cómo se indica al módulo el directorio donde se encuentra el archivo que necesita privilegios especiales, dándole la opción `+ExecCGI` lo que permite ejecutar las llamadas.

```
# MODULO CGI - EJECUTA SCRIPTS DE PYTHON A TRAVES DE APACHE
# ========================================================
<Directory /path_to_my_project>
```

```
    Order deny,allow
    Options +ExecCGI
    DirectoryIndex ejecutables.py
    AddHandler cgi-script .py
</Directory>
```

Fragmento 4.30: Activación del módulo CGI para la ejecución de scripts externos a Apache

4.4.4. Seguridad

El servidor en producción `Apache` cuenta con módulos de seguridad que permiten establecer la comunicación entre cliente y servidor de manera segura. La configuración del servidor en producción cuenta con dos archivos: el archivo con acceso a la aplicación a través del puerto 80 y el archivo donde se encuentran todos los módulos de configuración de los servidores `Django-1` y `Django-2` a través del puerto 443, puerto por defecto para peticiones `HTTP` seguras (`HTTPS`).

El primero de ellos, únicamente redirecciona todas las peticiones que se hacen al puerto 80, y las hace llegar al puerto de las peticiones HTTPS, el 443. Puede verse en el fragmento de código 4.31 lo explicado.

```
<VirtualHost *:80>
    ServerName www.site-app.com
    ServerAdmin admin@site

    # Redireccion a archivo SSL para encriptacion:
    Redirect permanent "/" "https:/url_to_site.es/"

</VirtualHost>
```

Fragmento 4.31: Configuración simple para el puerto 80

Con esta redirección se consigue forzar que todas las peticiones que vayan al puerto 80 (o las URLs que empiecen por el protocolo `'http'`) sean dirigidas al puerto que cuenta con el protocolo de seguridad para su encriptación `'https'`.

Las peticiones dirigidas a este archivo de configuración son encriptadas mediante el protocolo `TLS` (Transport Layer Security) ya que se encuentran dentro del módulo de seguridad `mod_ssl` como puede verse en el fragmento 4.32.

```
<IfModule mod_ssl.c>
    <VirtualHost _default_:443>

   [ . . . ]

    </VirtualHost>
</IfModule>
```

Fragmento 4.32: Configuración del archivo de Apache dirigido al puerto 443

Una vez la petición ha sido redirigida a este puerto, se van configurando en fragmentos denominados `módulos` cada una de las directivas que se deseen como se ha ido viendo a

lo largo de esta sección.

A pesar de que ya se tiene el sitio web con la seguridad activada, los navegadores no encuentran fiable el certificado que está devolviendo el servidor de `Apache` por no haber sido firmado por un *sitio de confianza*. Existen multitud de sitios web que ofrecen certificados de confianza. Para esta aplicación, se ha elegido la plataforma `Let's Encrypt`[1]. Esta plataforma es de código libre y permite de manera rápida y sencilla que el certificado sea de confianza para los navegadores, permitiendo extender las posibilidades de la aplicación web. Esta plataforma que proporciona el certificado, cuenta con patrocinadores como `Chrome`, `Cisco`, `Mozilla`, `Facebook` entre otras muchas, con el objetivo de promover la seguridad en las peticiones `HTTP` añadiendoles esta capa.

Dado que es un paquete que no viene instalado con el sistema operativo, el primer paso es obtenerlo del repositorio con la orden del fragmento 4.33, pasándole como uno de los argumentos el servidor en producción que estamos usando en este proyecto.

```
sudo apt-get install python-letsencrypt-apache
```

Fragmento 4.33: Orden para la instalación del paquete 'Letsencrypt'

Una vez instalado, mediante la orden del fragmento 4.34, se configura un certificado para el servidor en producción, pasandoselo como primer parámetro y el dominio donde se encuentra la aplicación, como último parámetro.

```
sudo letsencrypt --apache -d "dominio_de_la_aplicacion"
```

Fragmento 4.34: Configuración del paquete LetsEncrypt

Con esto ya tenemos configurado el servidor en producción para que su certificado sea válido. Por defecto, este certificado tiene un período de validez de 3 meses pero el propio paquete instalado contiene un comando para que se autorenueve cuando llegue la fecha de vencimiento como se ve en el fragmento 4.35.

```
sudo letsencrypt renew --dry-run --agree-tos
```

Fragmento 4.35: Orden para la autorenovación del certificado

Para comprobar que el certificado es válido, es decir, que es firmado por un sitio de confianza, los navegadores muestran en la ventana de introducción de URL un candado cerrado como los que se ven en la figura 4.7.

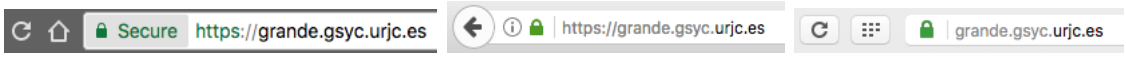


Figura 4.7: Indicador de página segura mostrada con un candado cerrado.

El uso del certificado de confianza es una garantía de que los datos están siendo cifrados en el origen hasta el destino, evitando que la intrusión de *'hombre en el medio'* pueda reemplazar a cualquiera de las partes y hacerle creer a la otra que es un usuario con acceso normal.

[1] `https://letsencrypt.org/`

4.5. Monitorización del servidor en producción

En este apartado se desglosa el archivo de monitorización que mantiene un control sobre el servidor y actúa en caso de que el servidor no se encuentre disponible. Este archivo se denomina `apache_daemon_checker.py` y se encuentra fuera del entorno de `Apache` como complemento al servidor en producción.

Este archivo hace una llamada al sistema operativo buscando el proceso de `Apache` para comprobar su actividad. Si éste se encuentra, el comportamiento del servidor es el correcto, pero si no está entre los procesos, lanzará la orden para reiniciar el servidor en producción y dejará una copia del error en el archivo `log` del sistema operativo con ruta (`/var/log/apache2/`).

El fragmento 4.36 de código contiene un bucle que, cada 60 segundos, comprueba la salida de un comando (`nmap -sT localhost`) que se le pasa al sistema operativo.

```
def run(self):
    i = 0
    while True:
        i += 1
        time.sleep(60)
        ps = commands.getoutput('nmap -sT localhost')
```

Fragmento 4.36: Bucle de comprobación de los puertos activos del servidor

Tal y como muestra el fragmento 4.37 si se cumple que los puertos del servidor 80, 443 y 8001 están en la salida del comando ejecutado, es decir, se encuentran activos, se guardará en el archivo `log` del sistema operativo la cadena: `'state OK'`. En caso contrario se reiniciará el servicio de `Apache` y del servidor de notificaciones y se dejará en el registro la cadena `'state ERROR'`.

```
[ . . . ]

if "80/tp" in ps and "443/tcp" in ps and "8001/tcp" in ps:
    # Todo OK. Se escribe en el archivo log:
    logger.info("state OK")
else:
    # Escritura en el archivo log
    logger.info("state ERROR")

    # Reinicio del servidor Apache
    os.system('sudo service apache2 restart')

    # Reinicio del servidor auxiliar para websocket
    os.system('cd /ruta_al_proyecto_django')
    os.system('python manage.py runserver 127.0.0.1:8001')
    os.system('cd')

[ . . . ]
```

Fragmento 4.37: Condiciones de comprobación ante el resultado

Este servicio en funcionamiento es gestionado por un fichero de arranque localizado en `/etc/init.d/` denominado `arranque_daemon.sh`, que permite realizar llamadas usando los métodos arrancar, parar o reiniciar el servicio de manera cómoda. Se muestra un extracto del fichero en el fragmento 4.38.

```
[ . . . ]

   start)
      echo "Starting server"
      python /usr/local/bin/apache_daemon_checker.py start
      ;;
   stop)
      echo "Stopping server"
      python /usr/local/bin/apache_daemon_checker.py stop
      ;;
   restart)
      echo "Restarting server"
      python /usr/local/bin/apache_daemon_checker.py restart
      ;;
   *)
      echo "Usage: /etc/init.d/arranque_daemon.sh {start|stop|restart}"
      exit 1
      ;;

[ . . . ]
```

Fragmento 4.38: Fichero de arranque del script

4.6. Backups

Otra herramienta que añade seguridad en los datos de la aplicación web de este TFG es la inclusión de un sistema de retorno y recuperación de los mismos. Para ello se ha contado con la herramienta que proporciona `MySQL` para la gestión de *Backups*. Se ha querido contar con dos tipos de configuración que trabajan de manera simultánea complementandose entre ellos, la *'copia completa'* y la *'copia incremental'*.

- **Copia Completa**

  Por un lado, se realiza una vez por semana una copia completa o total de la base de datos que incluye todas las tablas y contenidos de la base de datos especificada. Se programa una llamada a `MySQL` para que se exporte y se almacene en un directorio del sistema mediante la herramienta `Cron`, que viene por defecto instalada en los sitemas operativos `Linux` y permite establecer fechas concretas para la ejecución de comandos. Esta exportación, a su vez, es comprimida para el ahorro de espacio en disco. La orden ejecuta la exportación como se ve en el fragmento 4.39.

```
mysqldump -user -pass --opt dbdjango | gzip > "ruta_al_archivo_backup.sql.gz;
```

Fragmento 4.39: Exportación de la base de datos comprimida

  Esta copia se efectúa en el período de menor actividad de la aplicación, concretamente los Domingos a las 2:00 AM.

- **Copia incremental**

  Este procedimiento se consigue mediante el uso de los `binary logs` que son los archivos donde se encuentran almacenadas todas las transacciones de la base de datos. Para activar estos archivos hay que editar el archivo de configuración de la

base de datos `my.cnf` con ruta en `/etc/mysql/my.cnf` quedando el archivo como se ve en el siguiente fragmento:

```
log_bin = /var/log/mysql/mysql-bin.log
expire_logs_days = 10
```

Fragmento 4.40: Configuración de los Binary Logs

La primera línea indica la ruta donde se guardaran los logs, y la segunda línea muestra la duración de los archivos, que se ha dejado en 10 días.

Es necesario antes de una importación de la base de datos incremental congelar los `binary log` mediante el comando `flush-log` para guardar el estado de ese momento y comenzar uno nuevo del siguiente día, incrementándose en 1 el valor del nombre del archivo. El directorio de logs, en la primera configuración cuando se activan, tendrá los siguientes archivos `log`:

```
ls -al /var/log/mysql/
total 16
drwxr-s--- 2 mysql adm  4096 Mar  5 10:41 .
drwxr-xr-x 7 root  root 4096 Mar  5 09:52 ..
-rw-rw---- 1 mysql adm   106 Mar  5 10:41 mysql-bin.000001
-rw-rw---- 1 mysql adm    32 Mar  5 10:41 mysql-bin.index
```

Fragmento 4.41: Estado inicial del directorio donde se encuentran los Binary Logs

El archivo `mysql-bin.000001` es el log activo donde actualmente se estarán guardando todas las transacciones y el archivo `mysql-bin.index` es donde están todos los archivos `bin logs`.

Así pues, en caso de tener que recuperar el estado de la base de datos un día donde ha habido ciertas transacciones pero no están almacenadas en la copia completa, se podrían recuperar usando estos archivos y realizando la importación. Esta importación completaría el estado del último *backup* añadiendo además las transacciones hasta esa copia incremental. La importación se realiza de según muestra el fragmento 4.42.

```
mysqlbinlog mysql-bin.000001 | mysql -uroot -p backup
```

Fragmento 4.42: Importación de la base de datos incremental

De esta manera se garantiza una seguridad en los datos por si hubiera una pérdida masiva o algún error que provocara un reinicio de las tablas. Al final del día se guardan todas las transacciones en el archivo correspondiente y se genera uno nuevo que contiene las transacciones del siguiente día, quedando el directorio `/var/log/mysql/` como se ve en el fragmento 4.43.

```
ls -al /var/log/mysql/
```

```
total 16
drwxr-s--- 2 mysql adm  4096 Mar  5 10:41 .
drwxr-xr-x 7 root  root 4096 Mar  5 09:52 ..
-rw-rw---- 1 mysql adm   106 Mar  5 2:00 mysql-bin.000001
-rw-rw---- 1 mysql adm   106 Mar  6 2:00 mysql-bin.000002
-rw-rw---- 1 mysql adm    32 Mar  6 2:00 mysql-bin.index
```

Fragmento 4.43: Estado posterior a la importación de la BBDD incremental

Capítulo 5

Experimentos

En este capítulo se muestran los experimentos llevados a cabo para la validar el funcionamiento y probar servidor. Los primeros experimentos consisten en probar con un navegador web la cantidad de peticiones que se hacen, así como el tiempo que tarda el cliente en cargar todos los archivos, dando la página por construida. El objetivo es conocer el tiempo que el cliente espera para que la página sea utilizable para saber si el tamaño de los archivos de la página es el adecuado.

Se ha medido también el tiempo que tarda en recuperarse de una caída inesperada del servidor. Se ha comprobado igualmente la seguridad en las transacciones verificando la correcta encriptación de los datos. Con respecto a la base de datos, se ha recuperado el contenido de un estado anterior para comprobar el funcionamiento de los *backups*, así como un test *Benchmark* del rendimiento para conocer el número máximo de peticiones y usuarios que admite.

5.1. Hardware y software utilizado en las pruebas

La infraestructura hardware y software empleada en las pruebas ha sido:

- En el apartado **hardware** se han empleado dos ordenadores. Un ordenador portátil `MacBook Air` con un procesador `Intel(R)` i5 doble núcleo a 1.4 GHz con 4Gb de RAM para el prototipo de servidor con `Django`. Como servidor en producción se ha usado un ordenador de sobremesa con el sistema operativo `Ubuntu 16.04 LTS`, microprocesador `Intel(R) Core(TM)2 Quad CPU Q9300 @2.50GHz` con 4Gb de memoria RAM y con el servidor `Django` en su versión 1.9 (sobre `Python 2.7`) y `Apache2`.
- En el apartado del **software** se usa en el lado cliente cualquiera de los navegadores que cumpla con los estándares del `W3C` como son; `Google Chrome`, `Mozilla Firefox`, `Microsoft Edge` y `Opera`.

La conexión a Internet usada es doméstica, que consta de 32Mbps de bajada y 6.55 Mbps de subida.

5.2. Análisis de los tiempos de respuesta de la aplicación web.

Para estas pruebas, se usa el navegador `Google Chrome` como referencia para probar el número de solicitudes totales que recibe y el tiempo que tarda en la carga total de

los archivos para dos páginas, una de información sencilla y otra que requiere cierto procesamiento del servidor.

Figura 5.1: Navegadores Web

Esta prueba ayuda a contabilizar los archivos solicitados por el cliente y mide los tiempos de carga de las páginas percibidos por el cliente. El objetivo es canalizar si el rango de tiempos que tardan cliente y servidor en procesar las peticiones son los adecuados ante un uso típico de la aplicación.

El tiempo que tarda el cliente en tener la página operativa es la suma de tres tiempos. Uno de ellos es controlado por el cliente y la máquina donde está siendo ejecutada la aplicación, denominado *tiempo de procesamiento en cliente*. Otro valor depende de las operaciones que se realizan en el servidor, denominado *tiempo de procesamiento en el servidor*. El último valor no depende del uso ni de cómo esté hecha la aplicación ya que influyen un gran número de variables que no son controladas por ninguna de las partes sino por cómo esté diseñada la infraestructura de la red, es el *tiempo de transmisión*.

El tiempo que tarda el cliente en tener la página utilizable es la suma de los tres valores como se ve en el fragmento 5.1:

```
T. de Respuesta = T. de tranmisión + T. procesamiento cliente + T. procesamiento servidor
```

Fragmento 5.1: Tiempo de respuesta

Para la primera prueba se emplea la página inicial de la aplicación web, donde, en un primer momento, el cliente solicita todos los archivos de los que se compone la aplicación. Seguidamente se tramitará una solicitud a otra página contando con que el cliente ya tiene gran parte de los archivos, por lo que se espera unos tiempos de carga y transmisión de información menores. En la segunda prueba se accede al formulario del Parte de Obra donde están disponibles los campos para ser rellenos. En la tercera prueba en envía el Parte de Obra para contabilizar el tiempo en el que el cliente obtiene una respuesta del servidor una vez han sido guardados los datos. En la cuarta prueba, se accede al parte guardado para conocer los tiempos de acceso a la base de datos y la respuesta del servidor al cliente ante esa solicitud.

5.2.1. Acceso a la página de inicio

En la figura 5.2 puede verse una lista de los archivos solicitados al servidor cuando se accede a la *página inicial* de la aplicación así como el tiempo de carga.

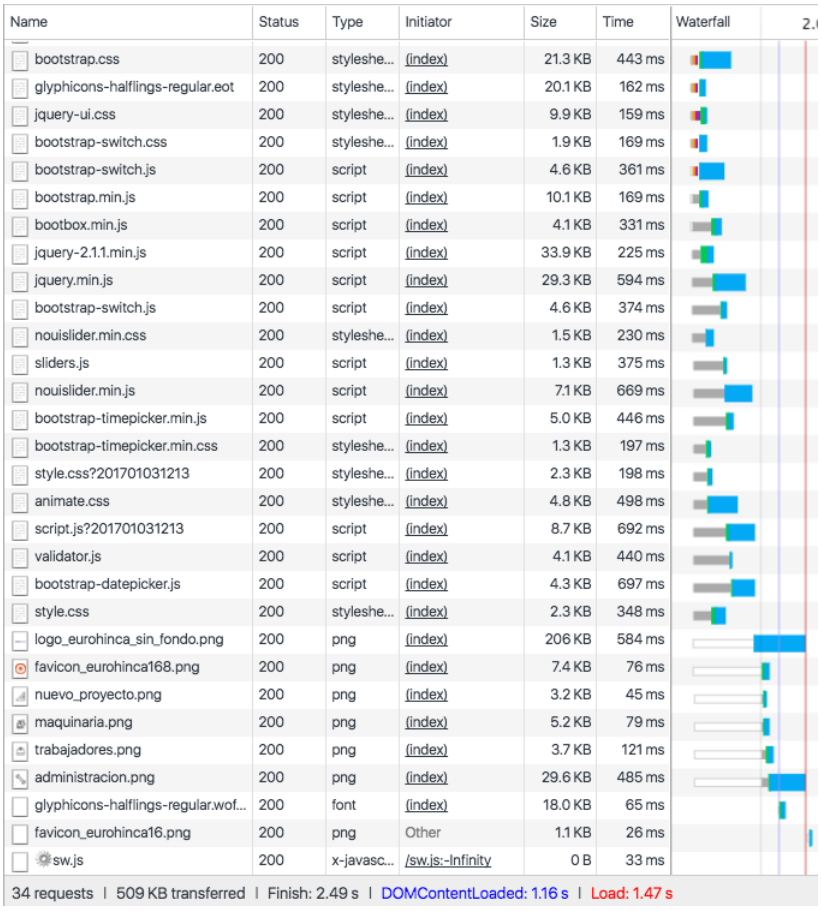


Figura 5.2: Cuadro de peticiones con el navegador Chrome en ordenador portátil

Los datos quedan resumidos en la tabla 5.1:

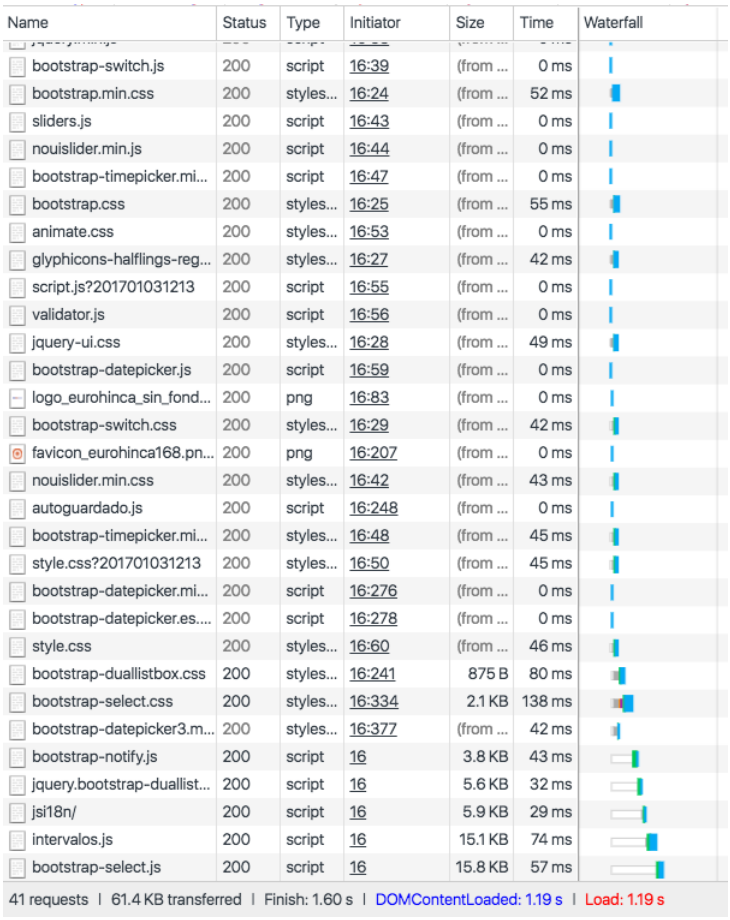
Solicitudes	34
Información Transferida	509 Kb
Tiempo total de renderización	2.49 segundos

Cuadro 5.1: Tabla de resultados para Google Chrome para ordenador portatil

Se concluye con estos datos que los tiempos de respuesta son los adecuados para una aplicación de este tipo cuando se solicitan todos los archivos en un primer acceso a la aplicación. Uno de los objetivos planteados en el capítulo 2 fue que la experiencia de usuario fuera la adecuada. Vemos con este ejemplo que ese punto se cumple, a pesar que es un acceso donde el cliente recibe *todos* los archivos.

5.2.2. Acceso a la página 'Parte de Obra'

En el siguiente ejemplo se realiza la petición al formulario de **'Parte de Obra'**. Buena parte de los archivos ya están almacenados en caché en el cliente por lo que el volumen de información recibida y los tiempos de carga deberían ser menores. La figura 5.3 muestra la solicitud a los archivos y los tiempos de carga:



Name	Status	Type	Initiator	Size	Time	Waterfall
bootstrap-switch.js	200	script	16:39	(from ...	0 ms	
bootstrap.min.css	200	styles...	16:24	(from ...	52 ms	
sliders.js	200	script	16:43	(from ...	0 ms	
nouislider.min.js	200	script	16:44	(from ...	0 ms	
bootstrap-timepicker.mi...	200	script	16:47	(from ...	0 ms	
bootstrap.css	200	styles...	16:25	(from ...	55 ms	
animate.css	200	styles...	16:53	(from ...	0 ms	
glyphicons-halflings-reg...	200	styles...	16:27	(from ...	42 ms	
script.js?201701031213	200	script	16:55	(from ...	0 ms	
validator.js	200	script	16:56	(from ...	0 ms	
jquery-ui.css	200	styles...	16:28	(from ...	49 ms	
bootstrap-datepicker.js	200	script	16:59	(from ...	0 ms	
logo_eurohinca_sin_fond...	200	png	16:83	(from ...	0 ms	
bootstrap-switch.css	200	styles...	16:29	(from ...	42 ms	
favicon_eurohinca168.pn...	200	png	16:207	(from ...	0 ms	
nouislider.min.css	200	styles...	16:42	(from ...	43 ms	
autoguardado.js	200	script	16:248	(from ...	0 ms	
bootstrap-timepicker.mi...	200	styles...	16:48	(from ...	45 ms	
style.css?201701031213	200	styles...	16:50	(from ...	45 ms	
bootstrap-datepicker.mi...	200	script	16:276	(from ...	0 ms	
bootstrap-datepicker.es....	200	script	16:278	(from ...	0 ms	
style.css	200	styles...	16:60	(from ...	46 ms	
bootstrap-duallistbox.css	200	styles...	16:241	875 B	80 ms	
bootstrap-select.css	200	styles...	16:334	2.1 KB	138 ms	
bootstrap-datepicker3.m...	200	styles...	16:377	(from ...	42 ms	
bootstrap-notify.js	200	script	16	3.8 KB	43 ms	
jquery.bootstrap-duallist...	200	script	16	5.6 KB	32 ms	
jsi18n/	200	script	16	5.9 KB	29 ms	
intervalos.js	200	script	16	15.1 KB	74 ms	
bootstrap-select.js	200	script	16	15.8 KB	57 ms	

41 requests | 61.4 KB transferred | Finish: 1.60 s | DOMContentLoaded: 1.19 s | Load: 1.19 s

Figura 5.3: Cuadro de peticiones con el navegador Chrome - Parte de Obra

Los datos quedan resumidos en la tabla 5.2:

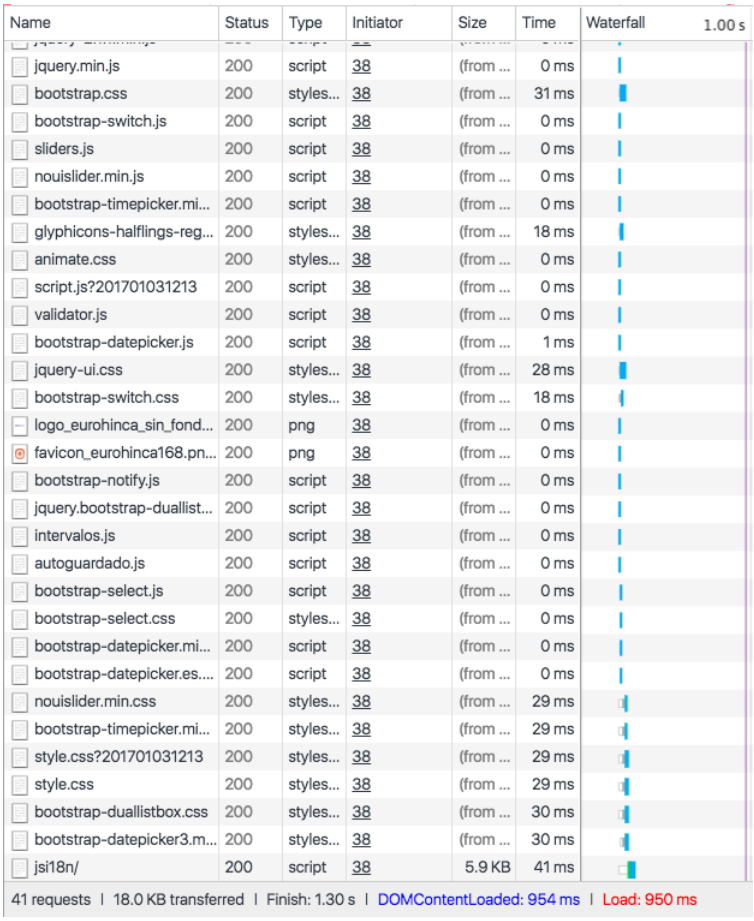
Solicitudes	41
Información Transferida	61.4Kb
Tiempo total de renderización	1.60 segundos

Cuadro 5.2: Tabla de resultados para Google Chrome - Parte de Obra

Puede comprobarse que, aunque el número de archivos es mayor, el cliente ya contiene gran cantidad de los documentos que necesita, por lo que la información transmitida entre cliente y servidor es menor. Además, el tiempo de carga es de 0.90 segundos menor que el acceso a la página inicial lo que ofrece mayor fluidez en la navegación.

5.2.3. Guardado de un Parte de Obra

Para este ejemplo se rellena el parte de obra solicitado en el punto anterior y se envía al servidor para que guarde esa información en la base de datos. El objetivo de esta prueba es comprobar el tiempo que tarda el servidor en guardar la información y devolver la página posterior al guardado. Al igual que los casos anteriores, la figura 5.4 muestra el resumen de los archivos solicitados, información transferida entre cliente y servidor y el tiempo de carga.



Name	Status	Type	Initiator	Size	Time	Waterfall 1.00 s
jquery.min.js	200	script	38	(from ...	0 ms	
bootstrap.css	200	styles...	38	(from ...	31 ms	
bootstrap-switch.js	200	script	38	(from ...	0 ms	
sliders.js	200	script	38	(from ...	0 ms	
nouislider.min.js	200	script	38	(from ...	0 ms	
bootstrap-timepicker.mi...	200	script	38	(from ...	0 ms	
glyphicons-halflings-reg...	200	styles...	38	(from ...	18 ms	
animate.css	200	styles...	38	(from ...	0 ms	
script.js?201701031213	200	script	38	(from ...	0 ms	
validator.js	200	script	38	(from ...	0 ms	
bootstrap-datepicker.js	200	script	38	(from ...	1 ms	
jquery-ui.css	200	styles...	38	(from ...	28 ms	
bootstrap-switch.css	200	styles...	38	(from ...	18 ms	
logo_eurohinca_sin_fond...	200	png	38	(from ...	0 ms	
favicon_eurohinca168.pn...	200	png	38	(from ...	0 ms	
bootstrap-notify.js	200	script	38	(from ...	0 ms	
jquery.bootstrap-duallist...	200	script	38	(from ...	0 ms	
intervalos.js	200	script	38	(from ...	0 ms	
autoguardado.js	200	script	38	(from ...	0 ms	
bootstrap-select.js	200	script	38	(from ...	0 ms	
bootstrap-select.css	200	styles...	38	(from ...	0 ms	
bootstrap-datepicker.mi...	200	script	38	(from ...	0 ms	
bootstrap-datepicker.es....	200	script	38	(from ...	0 ms	
nouislider.min.css	200	styles...	38	(from ...	29 ms	
bootstrap-timepicker.mi...	200	styles...	38	(from ...	29 ms	
style.css?201701031213	200	styles...	38	(from ...	29 ms	
style.css	200	styles...	38	(from ...	29 ms	
bootstrap-duallistbox.css	200	styles...	38	(from ...	30 ms	
bootstrap-datepicker3.m...	200	styles...	38	(from ...	30 ms	
jsi18n/	200	script	38	5.9 KB	41 ms	

41 requests | 18.0 KB transferred | Finish: 1.30 s | DOMContentLoaded: 954 ms | Load: 950 ms

Figura 5.4: Cuadro de peticiones con el navegador Chrome - Guardado del Parte de Obra

Los datos quedan resumidos en la tabla 5.3:

Solicitudes	41
Información Transferida	18 Kb
Tiempo total de renderización	1.30 segundos

Cuadro 5.3: Tabla de resultados para Google Chrome - Guardado de Parte de Obra

Puede verse que, como ocurría en el caso 5.2.2, el número de solicitudes es 41 pero la información intercambiada es mucho menor, únicamente 18kb. El tiempo que tarda el servidor en guardarlo, devolver la plantilla correspondiente y ser *renderizado* en el cliente es de 1.30 segundos, ligeramente menor al caso anterior.

5.2.4. Edición de un Parte de Obra

Por último, se va a solicitar un parte introducido al servidor para su edición. La intención de esta prueba es comprobar el tiempo que tarda en acceder al parte correspondiente de la base de datos (requiere más de una petición para conocer todos los datos como son trabajadores, maquinaria, proyecto y usuarios) y devolver la información.

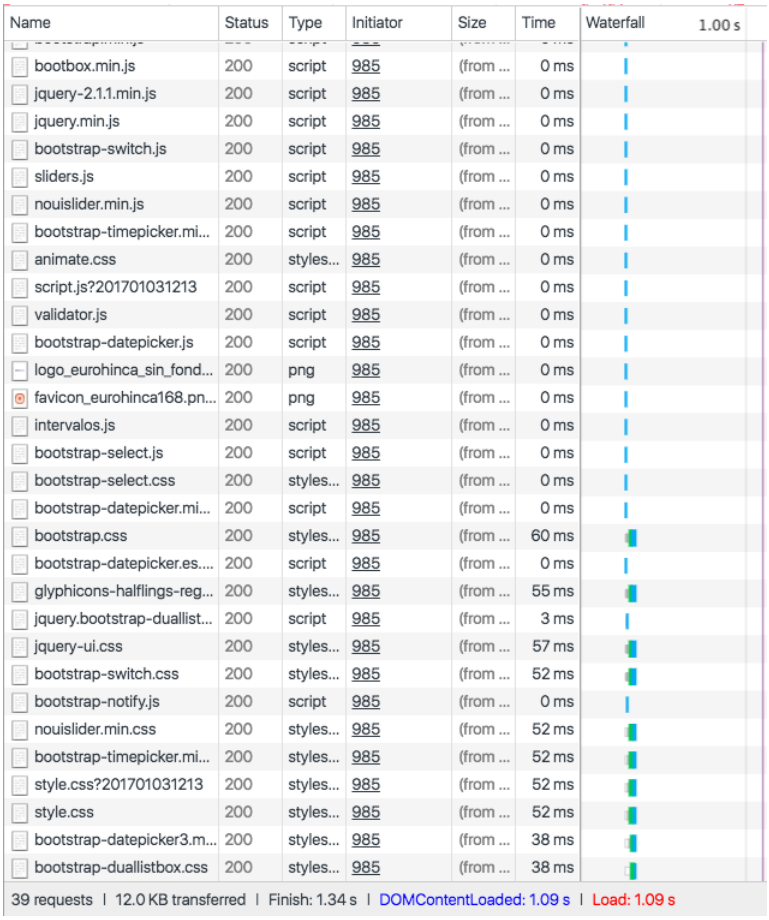


Figura 5.5: Cuadro de peticiones con el navegador Chrome - Edición del Parte de Obra

Los datos resumidos se presentan en la tabla 5.4:

Solicitudes	39
Información Transferida	12 Kb
Tiempo total de renderización	1.34 segundos

Cuadro 5.4: Tabla de resultados para Google Chrome - Edición de Parte de Obra

Esta última prueba demuestra que los tiempos de solicitud y respuesta por parte del servidor hacia la base de datos y su posterior respuesta al cliente son bajos. Esto se traduce en una buena eficiencia del servidor a la hora de buscar datos y devolverselo al cliente. Nótese que el cliente tiene que *renderizar* la plantilla, lo que conlleva un tiempo que no pertenece al servidor.

5.3. Prueba de rendimiento MySQL

Para poner a prueba el número de solicitudes que puede atender el servidor de la base de datos de MySQL, usaremos la herramienta que viene instalada en el paquete del servidor de la base de datos `MySQL` (a partir de la versión 5.1) y permite hacer pruebas de rendimiento, `mysqlslap`. Esta herramienta permite comparar el rendimiento del servidor ante cambios realizados en las bases de datos así como con *'n'* clientes interactuando con la base de datos, cierto tipo de consultas SQL, etc.

Para esta prueba de rendimiento, desde la consola del sistema operativo, se ejecuta la orden de `MySQL` (`mysqlslap`) pasándole los siguientes parámetros:

- `--auto-generate-sql`: Este parámetro genera automáticamente las consultas SQL al servidor mediante una serie de números aleatorios en distintas columnas y filas de las tablas.
- `--concurrency`: Dado que un único usuario accediendo al servicio no es una prueba válida si queremos estresar el servidor, añadimos más usuarios concurrentes en las pruebas y consultas SQL personalizadas.
- `--iterations`: Otro parámetro importante en un test de rendimiento es el número de veces que se realiza la prueba. Para ello usaremos el parámetro `iterations`.
- `--number-of-queries`: Por último, de los 50 usuarios concurrentes que realizarán peticiones, se configurará el test para que no todos hagan las mismas peticiones. Con este parámetro el número de consultas quedará repartido entre los usuarios que se especifiquen.
- **1- Test de rendimiento para 50 usuarios, 50 iteraciones y 50 peticiones**

```
mysqlslap -uuser -p --concurrency=50 --iterations=50 --number-of-queries=50 --auto-generate-sql
```

```
Benchmark
        Average number of seconds to run all queries: 0.018 seconds
        Minimum number of seconds to run all queries: 0.013 seconds
        Maximum number of seconds to run all queries: 0.032 seconds
        Number of clients running queries: 50
        Average number of queries per client: 1
```

  Dado que el número de *'queries'* es igual al número de usuarios, cada usuario realizará una única petición. Este test es muy simple pero da una idea del rendimiento cuando el número de usuarios es bajo. En las siguientes pruebas se incrementarán cada uno de los parámetros en busca de una configuración que la base de datos no pueda soportar.

- **2.- Test de rendimiento para 50 usuarios, 100 iteraciones y 500 peticiones**

```
mysqlslap -uuser -p --concurrency=50 --iterations=100 --number-of-queries=500 --auto-generate-
    sql
```

```
Benchmark
        Average number of seconds to run all queries: 0.098 seconds
        Minimum number of seconds to run all queries: 0.076 seconds
        Maximum number of seconds to run all queries: 0.155 seconds
        Number of clients running queries: 50
        Average number of queries per client: 10
```

Para este caso se han incrementado el número de peticiones para que cada usuario realice 10 además de repetirse la prueba 50 veces más. Esto incrementa notablemente los tiempos de acceso si son comparados con el caso anterior pero proporciona buena respuesta atendiendo a todas las solicitudes sin error.

- **3.- Test de rendimiento para 300 usuarios, 50 iteraciones y 300 peticiones**

```
mysqlslap -uuser -p --concurrency=300 --iterations=50 --number-of-queries=300 --auto-generate-
    sql
```

```
Benchmark
        Average number of seconds to run all queries: 0.164 seconds
        Minimum number of seconds to run all queries: 0.119 seconds
        Maximum number of seconds to run all queries: 0.329 seconds
        Number of clients running queries: 300
        Average number of queries per client: 1
```

Para un número de usuarios mayor al de la prueba anterior y con una única petición por usuario, la base de datos responde adecuadamente a todas las peticiones sin error.

- **4.- Test de rendimiento para 500 usuarios, 50 iteraciones y 2000 peticiones**

```
mysqlslap -uuser -p --concurrency=500 --iterations=50 --number-of-queries=2000 --auto-generate-
    sql
```

```
Benchmark
        Average number of seconds to run all queries: 0.760 seconds
        Minimum number of seconds to run all queries: 0.637 seconds
        Maximum number of seconds to run all queries: 1.044 seconds
        Number of clients running queries: 500
        Average number of queries per client: 4
```

Vemos cómo el número de peticiones por usuario es bajo pero se ha centrado esta prueba en incrementar los usuarios para ver la respuesta. Todas las solicitudes han sido atendidas sin error.

- **5.- Test de rendimiento para 500 usuarios, 50 iteraciones y 3000 peticiones**

```
mysqlslap -uuser -p --concurrency=500 --iterations=50 --number-of-queries=3000 --auto-generate-
    sql
```

```
Benchmark
        Average number of seconds to run all queries: 1.471 seconds
        Minimum number of seconds to run all queries: 1.187 seconds
        Maximum number of seconds to run all queries: 1.783 seconds
        Number of clients running queries: 500
        Average number of queries per client: 6
```

Durante esta prueba, el comando `mysqlslap` devuelve 23 errores de tipo `Too many conections` como se ve en el fragmento 5.2:

```
mysqlslap: Error when connecting to server: 1040 Too many connections
mysqlslap: Error when connecting to server: 1040 Too many connections
mysqlslap: Error when connecting to server: 1040 Too many connections
[ . . . ]
```

Fragmento 5.2: Errores en las consultas ante un gran número de peticiones

Este resultado indica que ha habido peticiones que no han podido ser atendidas. Para esta aplicación en concreto no se trata de un mal resultado, dado que la configuración de número de usuarios y peticiones que genera el error es muy poco probable. No obstante, si el uso fuera en aumento y existiera riesgo de que alguna solicitud no fuese

atendida, se procedería a configurar la base de datos para introducir un *balanceo de carga* mediante un segundo servidor de apoyo cuando se superan un número de usuarios/solicitudes.

5.4. Tolerancia a caídas

Esta prueba consiste en verificar el funcionamiento del script de shell que está en continua escucha del servicio de Apache. Este script se ejecuta cuando el servicio de Apache deja de estar entre los procesos del sistema operativo, por lo que entra en marcha para reiniciar los servicios y ponerlo de nuevo en activo.

Para comprobar el funcionamiento del servicio del servidor en producción, se usará la orden `ps aux`, filtrando por la secuencia `apache` o `www-data`, dando el resultado que se ve en el fragmento 5.3:

```
$ ps aux | grep apache
    www-data  3412  0.0  0.2 108984  4152 ?  S    07:35   0:00 /usr/sbin/apache2 -k start
    www-data  3413  0.0  5.1 750268 106192 ? Sl   07:35   0:11 /usr/sbin/apache2 -k start
    www-data  3414  0.0  0.9 2052136 20240 ? Sl   07:35   0:04 /usr/sbin/apache2 -k start
    www-data  3415  0.0  0.8 2051976 17040 ? Sl   07:35   0:04 /usr/sbin/apache2 -k start
```

Fragmento 5.3: Búsqueda del proceso de Apache

Se puede ver cómo el servicio `apache2` está activo con el usuario `www-data`.

Mediante la orden de la shell de `Linux` '`kill`' se fuerza el cierre del servicio como se ve en el fragmento 5.4:

```
$ kill -9 apache2
```

Fragmento 5.4: Cierre del proceso apache2

Dado que el script de monitorización comprueba el estado cada 60 segundos, cuando se cumpla el ciclo comprobará si los puertos 80, 443 y 8001 están activos pues es donde se encuentran las localizaciones de los servidores. Dado que hemos eliminado el proceso, el script relanzará de nuevo los servidores y dejará el mensaje del fragmento 5.5 en el archivo `servers_state.log` de la ruta `/var/log/`:

```
state ERROR - Restarting ...
```

Fragmento 5.5: Error en el servidor de Apache

Seguidamente se ejecutarán las órdenes que reinician los servicios como se ve en el fragmento 5.6:

```
\lstset{language=Bash, basicstyle=\scriptsize}

os.system('sudo service apache2 restart')
os.system('cd /home/path_to_project/ ; python manage.py runserver 127.0.0.1:8001 ; cd')
```

Fragmento 5.6: Reinicio del servidor Apache

Con el servicio reiniciado, se comprueba que todo vuelve a la normalidad revisando la última línea del archivo de registro `servers_state.log` donde, la cadena `state OK` del fichero, indicará que el servidor está funcionando de nuevo.

5.5. Seguridad - HTTPS

Para comprobar la calidad del certificado de seguridad creado se procede a realizar una auditoría que se aplica sobre un dominio en concreto mediante el uso de la plataforma `QualysSSL Labs`[1]. Este test de seguridad valora aspectos como el tipo de cifrado y la calidad del mismo, los intercambios que se realizan entre cliente y servidor de claves de seguridad (llaves) además de usar emuladores de sistemas operativos para comprobar la seguridad en navegadores anteriores a los actuales (retrocompatibilidad), etc, dando un resultado de aproximadamente 93,75 puntos sobre 100. Puede verse un resumen de la auditoría en la figura 5.6.

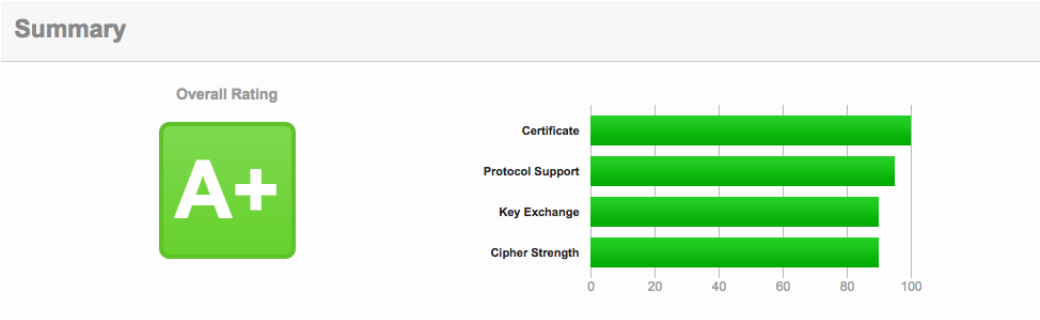


Figura 5.6: Resumen del estado de la seguridad en el dominio mediante el test de `https://www.ssllabs.com`

Las figuras 5.7, 5.8, 5.9, 5.10 y 5.11 muestran el informe completo del análisis de la página en el test de seguridad:

- Estado del certificado y características del mismo:

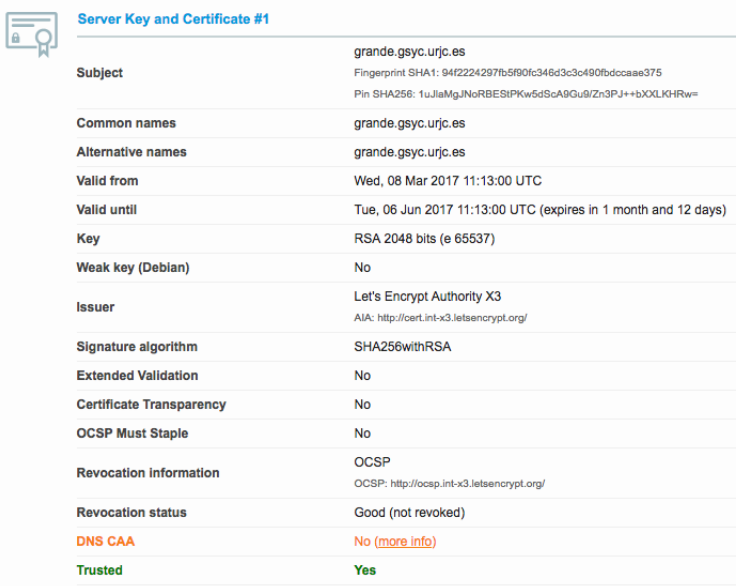


Figura 5.7: Estado de las claves del servidor y certificado

[1] fuente: `https://www.ssllabs.com/ssltest/`

- Certificados adicionales al que incluye por defecto:

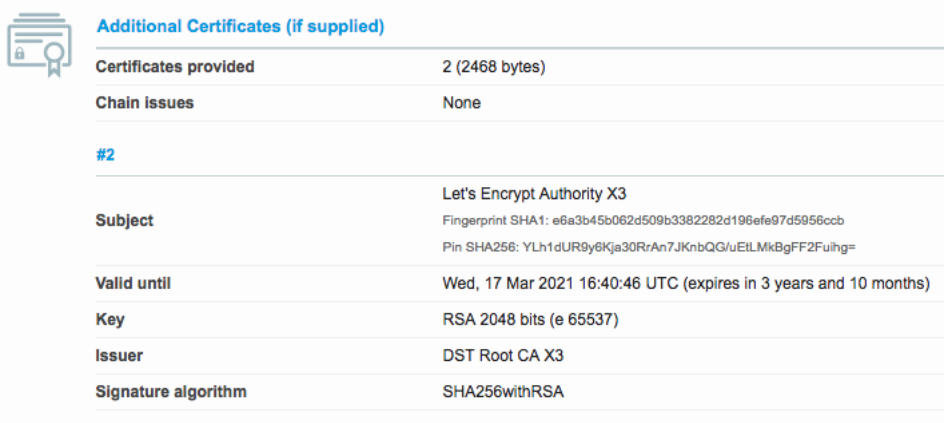


Figura 5.8: Certificados adicionales

- Protocolos TLS contenidos en este certificado:

Figura 5.9: Protocolos de seguridad contenidos en el certificado

- Conjunto de certificados contenidos:

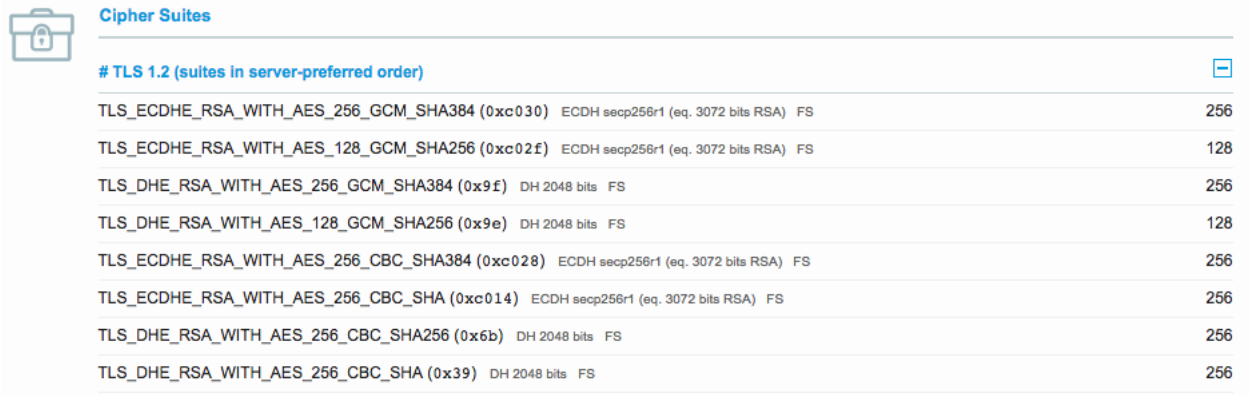
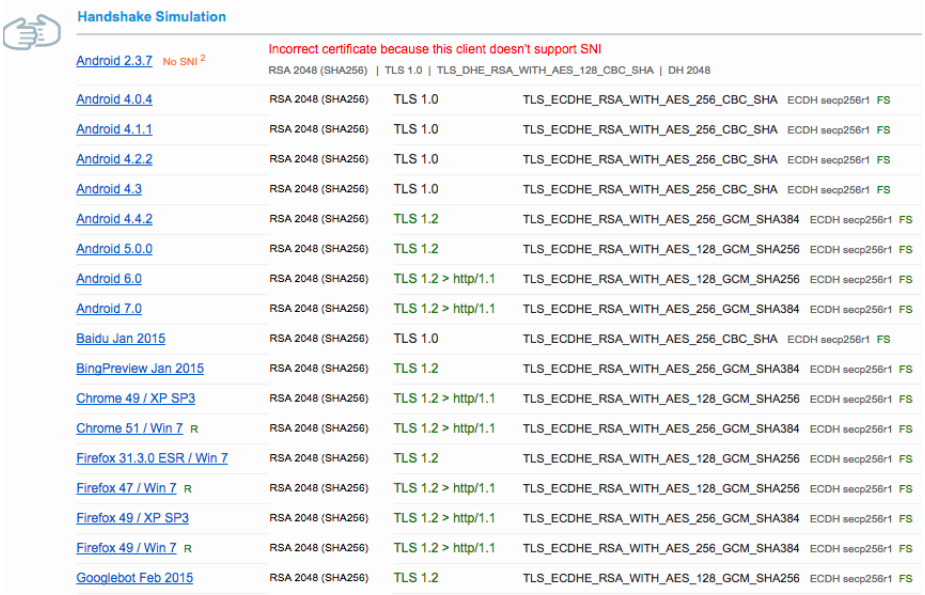


Figura 5.10: Conjuntos de cifrados

- Simulaciones de navegadores/sistemas operativos para ver las compatibilidades:



Handshake Simulation

Android 2.3.7 No SNI [2]	Incorrect certificate because this client doesn't support SNI RSA 2048 (SHA256) | TLS 1.0 | TLS_DHE_RSA_WITH_AES_128_CBC_SHA | DH 2048		
Android 4.0.4	RSA 2048 (SHA256)	TLS 1.0	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA ECDH secp256r1 FS
Android 4.1.1	RSA 2048 (SHA256)	TLS 1.0	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA ECDH secp256r1 FS
Android 4.2.2	RSA 2048 (SHA256)	TLS 1.0	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA ECDH secp256r1 FS
Android 4.3	RSA 2048 (SHA256)	TLS 1.0	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA ECDH secp256r1 FS
Android 4.4.2	RSA 2048 (SHA256)	TLS 1.2	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 ECDH secp256r1 FS
Android 5.0.0	RSA 2048 (SHA256)	TLS 1.2	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 ECDH secp256r1 FS
Android 6.0	RSA 2048 (SHA256)	TLS 1.2 > http/1.1	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 ECDH secp256r1 FS
Android 7.0	RSA 2048 (SHA256)	TLS 1.2 > http/1.1	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 ECDH secp256r1 FS
Baidu Jan 2015	RSA 2048 (SHA256)	TLS 1.0	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA ECDH secp256r1 FS
BingPreview Jan 2015	RSA 2048 (SHA256)	TLS 1.2	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 ECDH secp256r1 FS
Chrome 49 / XP SP3	RSA 2048 (SHA256)	TLS 1.2 > http/1.1	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 ECDH secp256r1 FS
Chrome 51 / Win 7 R	RSA 2048 (SHA256)	TLS 1.2 > http/1.1	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 ECDH secp256r1 FS
Firefox 31.3.0 ESR / Win 7	RSA 2048 (SHA256)	TLS 1.2	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 ECDH secp256r1 FS
Firefox 47 / Win 7 R	RSA 2048 (SHA256)	TLS 1.2 > http/1.1	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 ECDH secp256r1 FS
Firefox 49 / XP SP3	RSA 2048 (SHA256)	TLS 1.2 > http/1.1	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 ECDH secp256r1 FS
Firefox 49 / Win 7 R	RSA 2048 (SHA256)	TLS 1.2 > http/1.1	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 ECDH secp256r1 FS
Googlebot Feb 2015	RSA 2048 (SHA256)	TLS 1.2	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 ECDH secp256r1 FS

Figura 5.11: Simulación en navegadores/sistemas operativos

5.6. Backups

Para la gestión de *backups* se ha usado la herramienta que tiene incorporada MySQL. Mediante la orden del fragmento 5.7 se realiza una exportación completa de la base de datos.

```
mysqldump -u{user} -p --flush-log --master-data=2 {database} > "ruta_al_archivo/backup.sql.gz"
```

Fragmento 5.7: Exportación de la base de datos completa

Esta secuencia genera un archivo con extensión *.sql* que contiene toda la base de datos de la que se compone la aplicación. Para que ese almacenamiento de la base de datos sea óptimo desde el punto de vista del espacio en el disco, en el mismo momento que se ejecuta la exportación se comprime en formato `gz`.

Mediante la herramienta `Cron` se programa la exportación en la zona horaria donde menos actividad hay, en el caso de esta aplicación, por la noche. Se ha elegido las 2:00 AM como momento donde se realiza una *foto* de la base de datos. Por seguridad, en esa herramienta se hace una llamada a un archivo externo de otro directorio que contiene la órdenes para ejecutar la exportación.

Dado que este proceso no escalará cuando la base de datos tenga un alto contenido, se activa la opción *incremental*, que hace que la herramienta de *backup* sea más eficiente.

Como primer paso se activan los archivos binarios o `binary logs` editando el archivo `my.cnf` que se encuentra en la ruta `/etc/mysql/my.cnf` con las sentencias del fragmento 5.8.

```
log_bin = /var/log/mysql/mysql-bin.log
expire_logs_days = 7
```

Fragmento 5.8: Configuración del archivo Binary Logs

Estos comandos añadidos en el archivo indican dónde se guardarán los archivos binarios (por defecto en la ruta donde se encuentra `MySQL`) y los días que permanecerán antes de su eliminación.

En el comando anterior se pasó como parámetro la opción `flush log` que congelaba el estado de la base de datos para comenzar un nuevo registro. Además, se incluye `--master-data` que escribirá comentarios en el archivo exportado sobre las posiciones de los `logs`.

Si efectuamos alguna transacción a partir de la configuración anterior se verá reflejada en la carpeta mencionada anteriormente. `mysql-bin.000002` será el siguiente archivo con contenido transaccional al guardado anteriormente, como se ve en el fragmento 5.9.

```
ls -al /var/log/mysql/
total 16
drwxr-s--- 2 mysql adm  4096 Apr  24 10:41 .
drwxr-xr-x 7 root  root 4096 Apr  24 09:52 ..
-rw-rw---- 1 mysql adm   106 Apr  24 10:41 mysql-bin.000001
-rw-rw---- 1 mysql adm   106 Apr  24 10:41 mysql-bin.000002
-rw-rw---- 1 mysql adm    32 Apr  24 10:41 mysql-bin.index
```

Fragmento 5.9: Estado del directorio de los Binary Logs

Estos archivos (`mysql-bin.000001` y `mysql-bin.000002`) serán donde estarán almacenandas las transacciones. Por otro lado, el archivo `mysql-bin.index` es donde se encuentran todos los archivos `bin logs`. Con esto tendríamos suficiente para hacer una comprobación del funcionamiento de esta característica de `MySQL`:

- En primer lugar (contando con un Backup completo de la base de datos) tendríamos que eliminar la base de datos mediante el comando de `MySQL`: `"drop database;"`. Esto deja completamente limpia la base de datos de tablas existentes por lo que tendríamos que volver a crear la misma pero que se crearía vacía.

```
create database nombre_BBDD;
```

Fragmento 5.10: Creación de la base de datos vacía

- A continuación procederíamos a restaurar una versión completa (que se habría efectuado el Domingo anterior).

```
sudo gzip -dc < {backup}.sql.gz | mysql -u{user} -p {database}
```

Fragmento 5.11: Importación del backup completo

- Después de una versión completa, se procede a introducir los datos incrementales a ésta. Dado que, de los dos archivos `bin` que se encuentran en la carpeta, el `mysql-bin.000002` no ha sido congelado, se importarán los datos del archivo `mysql-bin.000001` mediante la siguiente orden.

```
sudo mysqlbinlog mysql-bin.000001 | mysql -u{user} -p {database}
```

Fragmento 5.12: Importación de la base de datos incremental

Una vez se efectúe la importación tendremos la base de datos en el estado deseado a partir del *backup* completo más la parte incremental que se necesite.

5.7. Localización e Internacionalización

Con estas pruebas se ha verificado que las herramientas de localización e internacionalización (`l10n` e `i18n`) entran en funcionamiento.

Para la primera prueba se accede a la aplicación desde un navegador con el idioma Inglés configurado por defecto. De esta manera, un acceso a la aplicación le dice al servidor `Django-1` que el navegador se encuentra configurado en el idioma Inglés. Esta configuración del cliente hará que el servidor devuelva el contenido de la página pero con el idioma cambiado. Puede verse la configuración del navegador en la figura 5.12.

Figura 5.12: Configuración del navegador. Acceso a la aplicación con idioma inglés como predeterminado.

El resultado de la carga de la página es el que se ve en la figura 5.13.

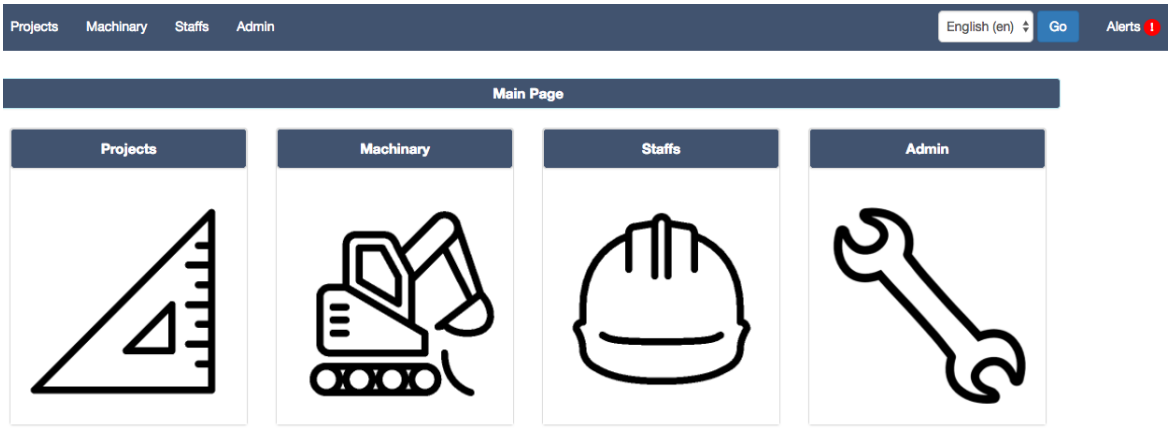


Figura 5.13: Acceso a la aplicación con el idioma Inglés configurado como predeterminado.

Si se vuelve a la configuración inicial y se establece el *Español* como idioma predeterminado, puede verse en la figura 5.14 como, automáticamente, cambia de nuevo al *Castellano*.

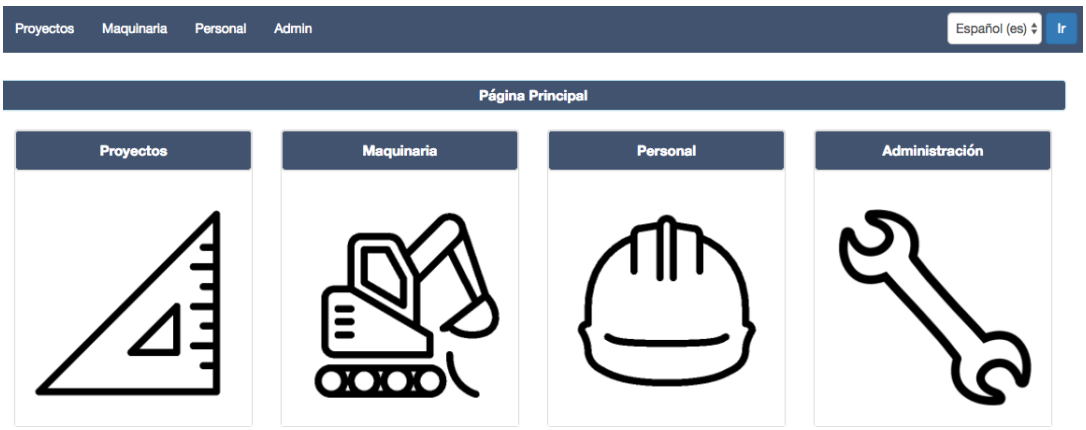


Figura 5.14: Acceso a la aplicación con el idioma Castellano configurado como predeterminado.

Para la segunda prueba se tiene el selector de idioma en la parte superior derecha de la página, en la barra de navegación, que permite alternar entre un idioma u otro simplemente clickando en uno de ellos.

Figura 5.15: Selector de idioma ubicado en la parte superior derecha.

El resultado es exáctamente el mismo que el visto en las figuras 5.13 y 5.14. Instantáneamente puede verse el cambio de idioma. Es una operación rápida y sencilla. La opción elegida se mantiene a lo largo de la navegación por una `cookie` que usa `Django` en cada interacción.

5.8. Alternativas probadas

Antes de la integración de una librería en la versión final de la aplicación han sido probadas varias alternativas que en principio cumplían con las espectativas buscadas pero que, al ir creciendo la aplicación, no han satisfecho las necesidades. Estas librerías son:

- **Librería para escribir en Excel - XLSWriter:**

  Esta librería fue la primera opción antes de usar `Openpyxl` por ser muy simple de escribir en el documento. El motivo que hizo cambiar a la actual fue porque no permite editar archivos ya creados y guardarlos, únicamente crear y guardar uno nuevo. Por ello el uso de las plantillas predefinidas por la empresa no era posible. Con la nueva librería ese problema quedó solventado.

- **Extensión para el uso de WebSockets - Redis:**

  Como primera opción, antes del uso de la librería `Channels` para `Django`, se usó como complemento para el uso de `WebSockets` el motor de búsqueda `Redis` por debajo del servidor en producción `Nginx`. Se descartó esta opción por dos motivos: el uso de `Redis` implicaba el despliegue de otro servidor en funcionamiento y requería de una estructura demasiado compleja para la sencillez del propósito.

- **Servidor en producción Nginx:**

  Se valoró el uso de `Nginx` como servidor en producción junto con `Apache`. La buena integración de éste último con `Django` a través del módulo `WSGI` para el despliegue y los módulos que completan la funcionalidad hizo que se eligiera `Apache` como servidor en producción final.

Capítulo 6

Conclusiones

En los capítulos anteriores se ha descrito el problema y se ha presentado una solución desarrollada justificando la elección de cada tipo de tecnología usada además de las pruebas realizadas. En este capítulo se analizarán las conclusiones finales a raíz del proyecto presentado así como posibles líneas futuras de desarrollo.

6.1. Conclusiones

Tras analizar el trabajo realizado se puede verificar que se ha cumplido con el objetivo principal. Se ha creado una aplicación web para uso interno de una empresa. A través de la interacción con un servidor, se ha ido guardando información necesaria para el tratamiento de partes de obra diarios en el que están involucrados muchos datos de distinto caracter que interaccionan entre sí.

Se han satisfecho todos los subobjetivos enmarcados en el capítulo 2 y que se desglosan a continuación:

- Se ha usado como base de datos relacional `MySQL` por ser de código libre y de las más usadas en este ámbito. Su gestión de tablas y la manera en que se relacionan hacen que el modelado de los datos sea más cómodo de gestionar. Además, cuenta con una herramienta interna de *backups* sencilla y que permite exportar las bases de datos de manera completa o incrementalmente.

  Tras sucesivas iteraciones y realimentación con la empresa, se ha alcanzado un modelo de datos adecuado a sus necesidades, incluyendo proyectos, partes de obra, intervalos de trabajo, maquinaria, usuarios y trabajadores.

- Se ha utilizado un entorno de desarrollo que se encuentra entre los más populares para aplicaciones web de este tipo, `Django`. Con él, se ha creado un servidor cuyo objetivo principal es la recolección de datos enfocado a la introducción de reportes diarios de obra. Este servidor permite guardar, editar, consultar y eliminar, mediante el uso de vistas; partes de obra, trabajadores, maquinaria y proyectos de manera rápida y sencilla.

- Se han materializado las funcionalidades avanzadas de la aplicación web. Se permite la exportación de los documentos en formato `Excel` y `pdf` mediante las librerías `openpyxl` y `LibreOffice` respectivamente. Se ha resuelto el sistema de notificaciones a través de la tecnología `WebSockets` mediante la librería `Channels` para `Django`. Esta permite enviar al cliente información que ocurre en tiempo real

en la base de datos sin que este tenga que refrescar la página. Además, mediante las herramientas de internacionalización `'l10n'` e `'i18n'` integradas en `Django`, se ha resuelto el soporte multilenguaje pudiendo, o bien cambiar el idioma en función de la necesidad o bien en función del idioma del navegador que accede a la aplicación web.

- Se ha usado `Apache` como servidor en producción por ser uno de los más usados para aplicaciones de este tipo y además por su buena integración con `Django` a través de la interfaz `WSGI`. El uso de módulos adicionales completan la funcionalidad de este servidor. Esto permite añadir una capa de seguridad (`TLS`) que establece comunicación segura entre cliente y servidor. Además, proporciona un módulo que permite ejecutar scripts externos al servidor `Django` y realizar llamadas al sistema operativo teniéndo un mayor control sobre cada uno.

Los requisitos especificados en el capítulo 2, también han sido satisfechos y se resumen a continuación:

- Como primer requisito se tenía que el TFG fuera desarrollado con tecnologías actuales. Este requisito se ha satisfecho usando `Django` como servidor para la gestión de las plantillas que se ofrecen al cliente y los datos que se almacenan en la base de datos. `Django` cuenta con un buen soporte por parte de los desarrolladores y la comunidad. Además, se ha usado `MySQL` como base de datos, siendo una de las más usadas por su buena escalabilidad, rapidez y buen soporte. Por último, `Apache` como servidor en producción, sigue siendo uno de los más populares en su ámbito.

- Mediante el uso de Plantillas y permisos de `Django`, se ha satisfecho el requisito del *acceso por roles*, adaptando a cada uno de los usuarios que accedan los proyectos asociados el contenido de parte de la plantilla que queda oculto por no disponer de un rol concreto. Esto se resuelve en la *renderización* de la plantilla que ocurre en el servidor por medio del *contexto* que permite almacenar en variables resultados de búsquedas en la base de datos, condiciones o datos concretos para que sean sustituidos en las variables de plantilla.

- El requisito de *consultar los datos en tiempo real* ha sido resuelto mediante el servidor `Django` como servidor y el uso del navegador como interfaz en el cliente. Esto permite que la aplicación web pueda ser usada en cualquier punto geográfico con acceso a Internet y con un dispositivo que cuente con un navegador que soporte los estándares `HTML5`, `CSS3` y `JavaScript`. Además, el contenido de la aplicación se ajusta automáticamente al dispositivo que se conecta mediante la responsividad ofrecida por librerías como `Bootstrap`.

- Unos de los requisitos más importantes para el uso de las aplicaciones web es la facilidad de uso e *intuitividad*. Gracias a los códigos de colores, simplicidad y orden de las páginas se permite almacenar gran contenido de datos de una manera más sencilla pudiéndose además, editar si fuera necesario. Este requisito aumenta la productividad de la empresa a la hora de introducir un Parte de Obra.

El software desarrollado se ha validado experimentalmente, con múltiples pruebas descritas en el capítulo 5 , verificando la actuación del servidor ante diferentes situaciones y se ha puesto a prueba la base de datos así como realizado copias de seguridad.

El proyecto completo consta de alrededor de 19.000 líneas de código. La distribución

que hace Django de los archivos y carpetas facilita la localización y contenido de cada uno separando los archivos de configuración de la lógica del servidor.

Este proyecto le ha servido a la empresa para dar un salto tecnológico en una aplicación adaptada a sus necesidades y que viene a resolver un problema de tiempos y productividad. La validación principal del software desarrollado en este TFG es que está siendo utilizado cada día por miembros de la empresa que introducen partes de obra diarios, maquinaria, personal, consulta de proyectos y exportación de los datos en formato Excel y pdf.

El desarrollo de esta aplicación web ha permitido adquirir conocimientos en diferentes ámbitos debido al caracter heterogéneo y múltiples tecnologías empleadas:

- Desde el punto de vista de la programación, se ha ampliado y perfeccionado el desarrollo y la configuración del framework Django además de mejorar en lenguajes como Python o Shell mediante scripts. Trabajar con el elevado número de vistas para construir toda la aplicación web requiere de buena organización y seguir el código de buenas prácticas en cuanto a estructura y documentación interna.
- Se han adquirido conocimientos en el despliegue de un servidor en producción, en este caso Apache además de la configuración de los módulos internos que extienden las posibilidades como la seguridad (TLS).
- Se han ampliado conocimientos del uso del sistema operativo Linux y sus directorios de configuración para los servidores Django y Apache.
- Se han adquirido conocimientos en la base de datos MySQL así como de la relación entre tablas y realización de *backups* automáticos usando el sistema operativo.
- Se han ampliado conocimientos de los estándares HTML5, CSS3, Javascript y Bootstrap.
- Se ha aprendido el uso de la tecnología WebSockets para su uso como gestor de notificaciones.
- Se han perfeccionado los conocimientos y usos de los sistemas de control de versiones, tanto git como GitLab.
- Se han adquirido habilidades de desarrollo de software en equipo, dentro de un proyecto de estas características donde la organización y trabajo con el tutor han sido un pilar muy importante.
- Se han adquirido los conocimientos necesarios para el desarrollo de esta memoria en formato Latex.

6.2. Trabajos Futuros

Este TFG se encarga fundamentalmente de la recolección de datos de manera fácil y rápida para los usuarios, pudiendo consultar y descargar la información desde cualquier sitio y dispositivo. A continuación se presentan características y funcionalidades que completarían el ecosistema de la aplicación. Son posibles líneas de continuación del trabajo desarrollado:

- **Tratamiento de datos voluminosos:** Un trabajo de inmediata implementación sería el tratamiento de los datos introducidos. Tendría como objetivo mostrar tendencias de cada grupo de proyectos de forma gráfica para preveer posibles averías o situaciones. Además, ofrecer resúmenes semanales, mensuales y anuales que puedan ser filtrados por proyectos, usuarios, maquinaria, etc.

  Con una nueva pestaña en la página inicial, se accedería a esta sección donde, mediante el uso de los filtros de `Django`, el cliente iría solicitando rangos de datos. Con ellos, el servidor generará los gráficos y tablas necesarios para conocer la información de proyectos, trabajadores, maquinarias, etc. Esta extensión completaría notablemente la aplicación cerrando el sistema de recolección, almacenamiento, *tratamiento y representación.*

- **Offline:** Dado que esta aplicación requiere de conexión a Internet para el almacenamiento de los datos, no siempre es posible disponer de conectividad con una red que permita el acceso. El deseable uso de un sistema `offline` permitiría al usuario navegar localmente por el contenido de la aplicación incluso en intervalos esporádicos sin conectividad. En la introducción de un Parte de Obra el navegador web estaría a la espera de un acceso a Internet que permitiera sincronizar con el servidor, enviándole los datos introducidos para ser almacenados.

- **Uso de la aplicación web como único portal:** Actualmente las bases de datos de `personal` y `maquinaria` contienen información simple. Una línea posible de trabajo sería integrar en la aplicación web las bases de datos oficiales de personal y maquinaria que se acceden actualmente con software independiente.

Bibliografía

[1] Proyecto JdeRobot http://jderobot.org/

[2] Django Project https://www.djangoproject.com/

[3] Página Digital Ocean para despliegue de Django con Apache https://www.digitalocean.com/community/tutorials/how-to-secure-apache-with-let-s-encrypt-on-ubuntu-16-04

[4] Librería OpenPyXL https://openpyxl.readthedocs.io/en/default/styles.html

[5] Letsencrypt para certificados de confianza https://letsencrypt.org/

[6] Certbot para configuración de Apache con SSL https://certbot.eff.org/#ubuntuxenial-apache

[7] Página de MySQL para backups incrementales https://dev.mysql.com/doc/refman/5.7/en/point-in-time-recovery.html

[8] Benchamark a la base de datos MySQL https://www.digitalocean.com/community/tutorials/how-to-measure-mysql-query-performance-with-mysqlslap

[9] Documentaciópn de Django para configuración de la internacionalización http://djangobook.com/using-translations-outside-views-templates/

[10] Libro de GitHub https://git-scm.com/book/es/v1/Empezando-Acerca-del-control-de-versiones

[11] Django Channels, Librería para WerSocket https://channels.readthedocs.io/en/stable/

[12] Bootstrap http://getbootstrap.com/

[13] Documentación de Apache https://httpd.apache.org/docs/2.4/configuring.html

[14] Evolución de la web http://www.evolutionoftheweb.com/#/evolution/day