



UNIVERSIDAD DE VALLADOLID

ESCUELA TÉCNICA SUPERIOR DE
INGENIEROS DE TELECOMUNICACIÓN

TRABAJO FIN DE MÁSTER

MÁSTER EN INGENIERÍA DE TELECOMUNICACIÓN

**DESARROLLO DE CUADROS DE MANDO (*DASHBOARD*)
PARA LA INDUSTRIA 4.0**

AUTOR:

D. Pablo Marina Boillos

Tutor:

Dr. D. Ignacio de Miguel Jiménez

Valladolid, 3 de julio de 2020

TÍTULO: **Desarrollo de Cuadros de Mando
(*dashboard*) para la Industria 4.0**

AUTOR: **D. Pablo Marina Boillos**

TUTORES: **Dr. D. Ignacio de Miguel Jiménez**

DEPARTAMENTO: **Teoría de la Señal y Comunicaciones e
Ingeniería Telemática**

TRIBUNAL

PRESIDENTE: **Dra. Dña. Patricia Fernández del
Reguero**

SECRETARIO: **Dr. D. Ramón J. Durán Barroso**

VOCAL: **Dr. D. Javier Manuel Aguiar Pérez**

PRESIDENTE SUPLENTE: **Dr. D. Lourdes Pelaz Montes**

SECRETARIO SUPLENTE: **Dr. D. Ramón de la Rosa Steinz**

VOCAL SUPLENTE: **Dr. D. Jaime Gómez Gil**

Donde hay voluntad, hay un camino.

RESUMEN

El auge del Big Data en la actualidad ha propiciado que cada vez más industrias gestionen su negocio con el apoyo de un cuadro de mandos o *dashboard*, que es una interfaz gráfica con información del rendimiento de sus procesos internos. Suele mostrarse en una página web, conectada a una base de datos, lo que permite modificar los parámetros de consulta para cambiar la visualización en el acto.

En este Trabajo Fin de Máster (TFM) se ha desarrollado un *dashboard* con la librería *dash*, de *Python*, para visualizar los Indicadores Clave de Rendimiento (KPIs, *Key Performance Indicators*) de los procesos de testeo realizados en una empresa ficticia para así facilitar la posterior toma de decisiones.

En primer lugar, se experimentó con las funciones de dicha librería por separado en una aplicación de prueba para familiarizarse con ellas. Después, se desarrolló el *dashboard* definitivo partiendo de una versión más sencilla elaborada anteriormente en el Grupo de Comunicaciones Ópticas de la Universidad de Valladolid y, por último, se realizó el despliegue del mismo en un servidor para dar soporte en el marco de una red privada.

Entre las aportaciones principales del TFM destaca (1) el habilitar la comunicación con una base de datos, mientras que la versión previa leía los datos de una hoja Excel, (2) la elaboración de un sistema para generar informes a partir de las visualizaciones mostradas en el cuadro de mando y la posibilidad de su impresión directa y su exportación a Microsoft Word, y (3) una funcionalidad para dar de alta y modificar información relativa a los operarios de la empresa y, por tanto, modificar la información almacenada en la base de datos en lugar de leerla únicamente.

PALABRAS CLAVE

Cuadro de mando, *dashboard*, base de datos, servidor, exportación

ABSTRACT

The popularity of Big Data nowadays, has led to an increase in the number of industries running their businesses relying on a dashboard, which is a graphical interface which provides information about the performance of their internal processes.

In this Master's Thesis, a dashboard has been developed by using the Python library "Dash", in order to visualize the Key Performance Indicators (KPIs) related to testing processes taking place in a fictional firm, so as to ease decision making in a future.

Firstly, some experiments were made with the isolated methods of the library in a sandbox to hone our skills with the tool. Next, the final dashboard was developed, based in an earlier and simpler version created within the Optical Communications Group of University of Valladolid and, finally, the application was deployed in a server to be used inside a private network.

Among the main contributions of this project are (1) to enable the communication with a database, whereas former version read data from an Excel spreadsheet, (2) to elaborate a system to generate reports, based on graphics displayed on the dashboard and the possibility of printing them directly and exporting them to Microsoft Word, and (3) a functionality to register and modify information related to workers in the enterprise and, hence, modifying the information stored in the database instead of just reading it.

KEYWORDS

Dashboard, database, server, exportation

AGRADECIMIENTOS

Quiero dar las gracias a mi tutor Nacho por haberme guiado a lo largo de la realización de todo este proyecto y sin el cual, no habría sido capaz de llevarlo a cabo, especialmente en este año tan difícil, que nos ha obligado a hacerlo todo de forma remota.

Gracias a Julián (quien diseñó la base de datos utilizada en el TFM), no solo por haberse compenetrado tan bien conmigo en el proyecto durante estos meses de trabajo, sino por todas las charlas que hemos tenido en el laboratorio y en los descansos, mientras hemos podido hacerlo.

Mis agradecimientos también van para el resto de componentes del Grupo de Comunicaciones Ópticas de la Universidad de Valladolid, por haberme proporcionado un ambiente de trabajo acogedor y agradable, y a todo el resto de mis compañeros de máster, por todas las vivencias que hemos tenido juntos en esta nueva etapa.

Gracias a mis padres por haber estado a mi lado, y al resto de mi familia, especialmente a mi prima pequeña Luna, por haber creído siempre en mí.

ÍNDICE GENERAL

RESUMEN	VII
PALABRAS CLAVE	VII
<i>ABSTRACT</i>	IX
<i>KEYWORDS</i>	IX
AGRADECIMIENTOS	XI
ÍNDICE GENERAL	XIII
ÍNDICE DE FIGURAS	XV
ÍNDICE DE TABLAS	XVII
1. INTRODUCCIÓN	1
1.1. USO DE LOS DASHBOARDS	1
1.2. PLANTEAMIENTO DEL CASO DE USO DEL TFM	1
1.3. OBJETIVOS DEL TFM	1
1.4. FASES DE DESARROLLO DEL TFM	2
1.5. ESTRUCTURA DE LA MEMORIA	3
2. INTRODUCCIÓN A LOS <i>DASHBOARDS</i> EN LA INDUSTRIA 4.0	5
3. DEMOSTRACIÓN DEL CUADRO DE MANDO DESARROLLADO	7
4. HERRAMIENTAS UTILIZADAS Y DECISIONES TOMADAS AL RESPECTO	19
4.1. ANACONDA	19
4.2. FLASK	19
4.3. DASH	20
4.4. SERVIDOR PARA DESPLEGAR LA APLICACIÓN	20
4.4.1. EL CICLO DE DESARROLLO DE SOFTWARE Y EL MODELO DTAP	20
4.4.2. SERVIDORES WSGI	22
4.5. MYSQL WORKBENCH	26
4.6. APACHE	26

4.7.	PAQUETE DE APLICACIONES PARA EL SERVIDOR LOCAL.....	27
5.	CONSIDERACIONES REALIZADAS EN EL DISEÑO Y EL DESARROLLO DE LA APLICACIÓN.....	31
5.1.	CONEXIÓN A LA BASE DE DATOS	31
5.2.	CREACIÓN Y EDICIÓN DE DATOS	31
5.3.	EXPORTACIÓN DE DATOS A UN INFORME	32
5.4.	IDENTIFICADORES DE SESIÓN	33
5.5.	PUESTA EN MARCHA DEL SERVIDOR.....	34
6.	CONCLUSIONES	35
6.1.	CUMPLIMIENTO DE LOS OBJETIVOS	35
6.2.	PRINCIPALES IDEAS EXTRAÍDAS.....	35
6.3.	LIMITACIONES Y LÍNEAS FUTURAS	36
7.	BIBLIOGRAFÍA.....	39
APÉNDICE A.	ABREVIATURAS Y ACRÓNIMOS	41
APÉNDICE B.	MATERIAL UTILIZADO	43
APÉNDICE C.	PRESUPUESTO	45

ÍNDICE DE FIGURAS

Figura 1 Portada de la aplicación	8
Figura 2 Desglose por años para un rango determinado en Análisis temporal	9
Figura 3 Desglose por meses para un año en concreto en Análisis temporal	9
Figura 4 Vista de la pestaña Histórico de productos.....	10
Figura 5 Vista de la pestaña Análisis de productos	11
Figura 6 Vista de la pestaña Análisis de lotes.....	12
Figura 7 Número de tests y nivel de defectos medio para cada operario	13
Figura 8 Cada punto representa un test realizado por un operario en un momento concreto. Si se amplía el eje horizontal lo suficiente, puede verse la fecha y la hora de cada test.	13
Figura 9 Listar operarios. Se indica el identificador, el nombre y su estado en ese momento. .	14
Figura 10 Añadir operario. Si ya existe ese nombre, pedirá confirmación para añadirlo.....	14
Figura 11 Cambiar estado de un operario. Para ello, buscarle y pulsar el botón "Cambiar"	14
Figura 12 Cambiar estado de un operario. Puede volver a cambiarse el estado de un mismo operario las veces seguidas que se quiera.	15
Figura 13 Vista de la pestaña Acerca de.....	15
Figura 14 Botones de "Exportar", "Imprimir" y "Descargar"	16
Figura 15 Cuadro de impresión del dashboard	16
Figura 16 Vista previa antes de imprimir	17
Figura 17 Ciclo SDLC (Rather & Bhatnagar, 2016).....	21
Figura 18 DTAP combinado con Agile (Offshore Software Solutions, 2019)	21
Figura 19 Un servidor WSGI invoca a objetos llamables en la aplicación escrita en Python (Makai, 2019)	23
Figura 20 El estándar WSGI propone unas reglas de comunicación comunes entre el servidor web y el framework de la aplicación, sean cual sean estos (Alonso, 2019).....	23
Figura 21 Menú principal de WampServer	27
Figura 22 Acceso a la configuración y gestión de cada módulo.....	28
Figura 23 Selección de los componentes que se desean instalar en XAMPP Control Panel.....	28
Figura 24 Interfaz gráfica de XAMPP Control Panel	29

ÍNDICE DE TABLAS

Tabla 1 Comparativa entre servidores WSGI	25
Tabla 2 Resumen de las características de WampServer y XAMPP Control Panel	29

1. INTRODUCCIÓN

1. INTRODUCCIÓN

1.1. USO DE LOS DASHBOARDS

En los últimos años, el avance de la tecnología ha permitido que sean cada vez más empresas las que usen un cuadro de mando o *dashboard* como herramienta de gestión. Un *dashboard* comprende una interfaz, generalmente visual, donde se muestra información acerca del negocio y a través de la cual se monitorizan distintas variables de los procesos internos, que permitan tomar decisiones para optimizarlos y obtener el máximo rendimiento de ellos. En este Trabajo Fin de Máster (TFM) se desarrolla un *dashboard* para una empresa ficticia que realiza tests de calidad de productos propios, así como externos, y que muestra por pantalla los resultados de dichos tests e información asociada.

1.2. PLANTEAMIENTO DEL CASO DE USO DEL TFM

La empresa ficticia ACME elabora una serie de productos manufacturados y realiza validaciones de la calidad del género propio, así como el de otros proveedores externos. Nuestro objetivo es elaborar un cuadro de mando que muestre los resultados e información relativa a los tests realizados y que permita, a su vez, poder exportar estos datos a un fichero *Word* y ofrecer la posibilidad de imprimirlos o descargarlos para volverlos a analizar más tarde o ser remitidos a otras personas o entidades. Se tomará como punto de partida una base de datos relacional, a la que se conectará la aplicación para consultar los datos solicitados por el usuario, empleando para ello el Lenguaje de Consulta Estructurada (SQL, *Structured Query Language*).

1.3. OBJETIVOS DEL TFM

El objetivo global de este TFM consiste en desarrollar un cuadro de mando (*dashboard*) para llevar a cabo la visualización de distintos tipos de información que muestren los Indicadores Claves de Rendimiento (KPIs, *Key Performance Indicators*) de una empresa. El punto de partida es una versión básica de un cuadro de mando, desarrollada en el seno del grupo de investigación en el que se ha realizado el TFM, el Grupo de Comunicaciones Ópticas (GCO) de la Universidad de Valladolid (UVa). Esta versión inicial representaba un conjunto de datos obtenidos desde una hoja de cálculo Excel. Así, pues los principales objetivos del TFM son:

- Leer los datos almacenados en una base de datos y necesarios para su representación en el cuadro de mando.
- Permitir la creación y actualización de datos existentes en la base de datos desde el cuadro de mando.
- Permitir la exportación de documentos con las tablas y/o gráficas deseadas mediante la implementación en la interfaz de alguna funcionalidad pensada para este fin.

1. INTRODUCCIÓN

- Permitir realizar una impresión de las distintas secciones del cuadro de mando que sea más elegante que la que se obtiene dando simplemente al botón de imprimir del navegador.
- Establecer un procedimiento que permita desplegar la aplicación en un servidor local para su acceso de forma remota desde dentro de una red privada de la empresa.

1.4. FASES DE DESARROLLO DEL TFM

Las fases que se han seguido para la consecución de los objetivos anteriores es la siguiente:

- Fase de familiarización con los objetivos el proyecto, el caso de uso y las siguientes herramientas básicas del proyecto:
 - *Anaconda*
 - *Dash*
 - *MySQL*
 - *Flask*
- Fase de desarrollo de un cajón de arena o *sandbox*:
 - Aplicación de los conocimientos adquiridos en la fase previa a una aplicación de ensayo.
 - Implementación y testeo de una conexión de la aplicación a una base en *MySQL* conteniendo datos de prueba.
 - Discusión acerca de las consideraciones de diseño necesarias para la consecución de los objetivos estipulados.
 - Implementación y testeo de las funcionalidades producto de las consideraciones de diseño en una máquina local.
- Fase de despliegue del *sandbox*:
 - Familiarización con el servidor web *Apache*.
 - Comparativa, elección y puesta en marcha de las herramientas necesarias para desplegar el *sandbox*
 - Testeo del *sandbox* montado en *Apache* y de la conexión remota desde otros terminales electrónicos.
- Fase de desarrollo del *dashboard*:
 - Implementación y testeo de una conexión de la aplicación a la base en *MySQL* con los datos de consulta.
 - Implementación y testeo de las funcionalidades discutidas a lo largo de las fases anteriores.
- Fase de despliegue del *dashboard*:
 - Puesta en marcha del servidor ya con la nueva aplicación.
- Fase de revisión y conclusiones:
 - Comprobación sobre la aplicación final acerca de la trazabilidad de cada uno de los objetivos establecidos.
- Fase de elaboración de la memoria del TFM.

1. INTRODUCCIÓN

1.5. ESTRUCTURA DE LA MEMORIA

El presente documento de este TFM está estructurado de la siguiente forma. El Capítulo 1 realiza una breve introducción al uso de los *dashboards* en el mundo empresarial. El Capítulo 2 muestra la aplicación final desarrollada, explicando sus principales funcionalidades, para que así resulte más sencillo poner en contexto las explicaciones técnicas que se dan en los capítulos posteriores. El Capítulo 4 realiza una discusión detallada de todas las herramientas que se han necesitado para llevar a cabo el proyecto, así como algunas comparativas de alternativas disponibles para ciertos tipos de herramientas y su decisión justificada. El Capítulo 5 explica desde un punto de vista técnico cómo se ha llevado a cabo el cumplimiento de algunos de los objetivos planteados anteriormente, así como otras cuestiones relacionadas de interés, y el Capítulo 6 concluye esta memoria.

Asimismo, se incluyen tres apéndices con información complementaria sobre el proyecto. El Apéndice A recoge todas las abreviaturas y acrónimos usados en esta memoria y su significado, el Apéndice B recoge el material utilizado en el proyecto y el Apéndice C recoge el presupuesto estimado para llevar a cabo el proyecto.

1. INTRODUCCIÓN

2. INTRODUCCIÓN A LOS *DASHBOARDS* EN LA INDUSTRIA 4.0

La Industria 4.0 está basada fundamentalmente en la integración de computación, comunicación y control, y en el análisis de grandes volúmenes de datos (Big Data). Así pues, existen varios facilitadores de la Industria 4.0, entre los que se encuentra Internet, como infraestructura de comunicaciones, el paradigma de Internet de las Cosas, la computación en la nube, el Big Data, la inteligencia artificial, la robótica y la interacción humano-máquina (Aceto, Persico, Pescapé, & Member, 2019; Alcácer & Cruz-Machado, 2019; Raptis, Passarella, & Conti, 2019).

Este gran avance de la tecnología en los últimos años ha propiciado un cambio de paradigma en el mundo empresarial, donde se está produciendo un aumento exponencial de información generada relativa a la empresa y a sus procesos, convirtiéndose el dato, de esta forma, en su mayor activo y en su principal motor económico (Popkova, Ragulina, & Bogoviz, 2018)

La información que genera un negocio se suele analizar para la extracción de conclusiones que conlleven a mejorar el rendimiento de los procesos internos de la empresa, en beneficio de la misma. A este conjunto de técnicas de análisis de datos se le denomina Inteligencia de Negocio (BI, *Business Intelligence*).

Sin embargo, este estudio analítico se vuelve cada vez más complicado, a medida que aumenta el volumen de las cifras a analizar, lo que podría desviar el foco de atención a otros aspectos no pertenecientes a los objetivos primarios del negocio y, por ende, afectar negativamente a la toma de decisiones eficaces (Bradea, Sabău-Popa, & Boloş, 2016).

Un cuadro de mando o *dashboard* puede simplificar esta tarea. Consiste en una interfaz visual, a través de la cual se pueden mostrar distintos gráficos y tablas con información relevante del negocio. En función de los objetivos planteados, el *dashboard* puede ayudar a la monitorización de aquellas métricas o Indicadores Clave de Rendimiento, KPI (*Key Performance Indicators*), que sean clave para la empresa para poder detectar fácilmente y corregir problemas que surjan en el proceso de producción. Además de proporcionar una visión general del negocio, permite centrarnos en aquellos verdaderos aspectos relevantes de fabricación, que conduzcan a la introducción de cambios que optimicen los procesos inherentes a la empresa (Bradea et al., 2016; Shamsuzzoha, Hao, Helo, & Khadem, 2014).

La importación de los datos en crudo objeto de visualización puede realizarse mediante un fichero Excel, que se suele cargar completo dentro de la aplicación para posteriormente llevar a cabo la visualización por la interfaz. Esto puede ser un método eficaz con un conjunto de datos relativamente sencillo y pequeño. No obstante, si tenemos un fichero de extensión *xls* o *xlsx* bastante voluminoso, al importarlo de forma íntegra en nuestro código, puede afectar de forma negativa a los retardos del servidor y a la calidad de servicio.

2. INTRODUCCIÓN A LOS *DASHBOARDS* EN LA INDUSTRIA 4.0

En este TFM se desarrolla un *dashboard* que se conecta a una base de datos implementada en un servidor *SQL*, para realizar esas mismas consultas, pero con carga parcial, aliviando de esta forma el coste computacional de la aplicación.

3. DEMOSTRACIÓN DEL CUADRO DE MANDO DESARROLLADO

3. DEMOSTRACIÓN DEL CUADRO DE MANDO DESARROLLADO

Este apartado pretende introducir mínimamente al usuario en el uso del *dashboard* desarrollado en este proyecto. Se recuerda brevemente el caso de uso central.

La empresa ACME fabrica una serie de productos manufacturados y evalúa la calidad de productos tanto propios como elaborados por otros fabricantes. La empresa tiene diversas factorías situadas en distintos lugares y en cada factoría tiene diversas máquinas para realizar los tests. El cuadro de mando muestra la información relativa a los tests realizados por una máquina concreta seleccionada por el usuario de la aplicación.

Un producto elaborado por una empresa (por ejemplo, tornillos), tendrá asociado un código (ej., A1) y una serie de lotes (ej. el lote 12AB es un conjunto de 5000 tornillos elaborados en un intervalo de tiempo determinado y el lote 34CD es otro conjunto de 3000 tornillos elaborados en otro intervalo distinto de tiempo). La empresa usa una máquina cualquiera de la empresa para realizar un test de calidad de un lote de un producto con un protocolo determinado entre cinco distintos (ej. protocolo 3) y, en el caso del protocolo 1, con diez variantes distintas (ej. variante 5). A veces puede repetirse un test para un mismo lote, pudiéndose usar además distintos protocolos y/o variantes en cada una de esas pruebas. El resultado de cada test es una medida del nivel de defectos de ese lote que varía entre 0 y 45 (cuanto más bajo el nivel de defectos, mayor calidad), y una descripción nominal que indica si la calidad es ALTA, MEDIA o BAJA. Para ello, se utilizan dos umbrales determinados que vienen dados por la configuración de las máquinas, entre 4 posibles. Por ejemplo, si los umbrales son 12 y 30 y el nivel de defectos es 9, al ser menor que 12, tenemos calidad ALTA. Si el nivel de defectos es 18, al estar entre 12 y 30, la calidad es MEDIA y si el nivel de defectos es 40, al ser mayor que 30, la calidad es BAJA.

El desarrollo del presente epígrafe parte del supuesto de que el servidor web que contiene la aplicación está activado y de que el usuario utiliza un dispositivo conectado a la misma red privada que el servidor.

Para comenzar a utilizar el cuadro de mandos, será necesario abrir un navegador e introducir la URL (*Uniform Resource Locator*, Localizador de Recursos Uniforme) que da acceso al mismo, en este caso: `http://acme:8081/acme.html`

Tras ello, el usuario se encontrará con la portada de la aplicación, que presenta el aspecto de la Figura 1.

3. DEMOSTRACIÓN DEL CUADRO DE MANDO DESARROLLADO



Figura 1 Portada de la aplicación

Tras pulsar en el botón “Inicio”, el usuario entrará en el cuadro de mando, el cual está dividido en varias pestañas, de las que a continuación se expone el contenido de cada una. Salvo que se indique lo contrario, los datos que se muestran están asociados a una factoría y a una máquina en concreto que deberá elegir el usuario. Tras decidir la factoría, el usuario deberá escoger una máquina por su identificador entre todas las de dicha fábrica. Por último, deberá pulsar el botón “Refrescar” para que las gráficas y tablas de esa pestaña se actualicen (Figura 2).

Como podemos observar también, hay un botón rojo en la parte superior derecha del cuadro de mandos, que sirve para abandonarlo y volver a la vista de la Figura 1. El usuario puede salir del *dashboard* desde cualquiera de sus pestañas.

La pestaña “Análisis temporal”, muestra el promedio de tests, lotes analizados y productos analizados por año y por mes, asociados a una máquina y una factoría concreta. El usuario puede seleccionar el rango de años que desea visualizar y el año en concreto del cual desea ver el desglose por meses (Figuras Figura 2 y Figura 3).

3. DEMOSTRACIÓN DEL CUADRO DE MANDO DESARROLLADO

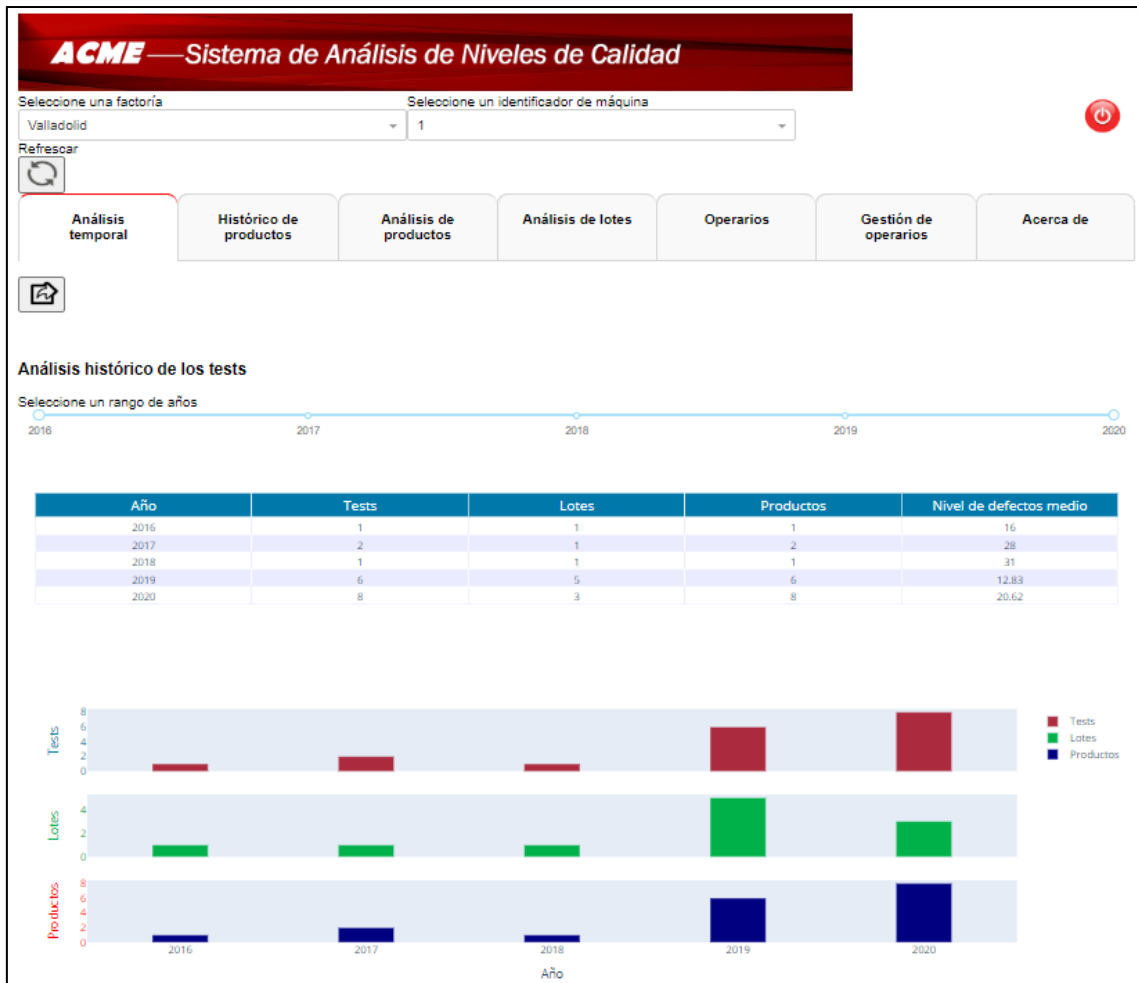


Figura 2 Desglose por años para un rango determinado en Análisis temporal



Figura 3 Desglose por meses para un año en concreto en Análisis temporal

3. DEMOSTRACIÓN DEL CUADRO DE MANDO DESARROLLADO

La pestaña “Histórico de Productos” muestra el número de tests y la distribución del nivel de defectos por producto para un rango de fechas concreto que deberá elegir el usuario (Figura 4).



Figura 4 Vista de la pestaña Histórico de productos

La pestaña “Análisis de productos” muestra el nivel de defectos medio y el número de tests por cada lote y en un rango de fechas determinado, ambos seleccionados por el usuario. Para cada lote en la gráfica de nivel de defectos medio, aparecen superpuestos los umbrales de la configuración usada en el test más reciente para esa combinación producto-lote. Como se puede observar en la Figura 5, la parte inferior de cada columna es verde; la intermedia, amarillenta; y la superior, rojiza, ya que cuanto menor nivel de defectos, mejor.

3. DEMOSTRACIÓN DEL CUADRO DE MANDO DESARROLLADO



Figura 5 Vista de la pestaña Análisis de productos

La pestaña “Análisis de lotes” muestra información acerca del número de lotes, de tests y el nivel de defectos medio para cada uno de los productos que tengan un lote con el identificador seleccionado. Debe tenerse en cuenta que, en muchos casos, los lotes de distintos productos utilizarán notaciones diferentes, pero puede darse el caso que usen la misma notación y, por ejemplo, haya un lote ‘12AB’ de tornillos y un lote ‘12AB’ de tuercas, así que es necesario tener en cuenta tanto el nombre del lote como el identificador del producto para representar la información de forma correcta.

Para el nombre de lote escogido, se muestra el número de tests y el nivel de defectos medio entre todos ellos, desglosando además el valor concreto de nivel de defectos, los umbrales elegidos, el protocolo usado y la variante si la hubiera para cada test.

A esta pestaña podemos acceder a ella de dos maneras. Una de ellas es pulsando sobre el rótulo de la misma, y otra, haciendo clic en la barra de uno de los lotes de cualquiera de las dos gráficas de la pestaña anterior. La diferencia está en que en el primer caso, el usuario deberá introducir manualmente el nombre de lote que desea visualizar, mientras que en el segundo, el lote asociado a la barra pulsada se rellenará automáticamente y la información se mostrará de forma directa (ej., supongamos que en la Figura 5 hemos pulsado la barra del lote ‘34CD’, y el resultado que obtendríamos sería el de la Figura 6).

3. DEMOSTRACIÓN DEL CUADRO DE MANDO DESARROLLADO

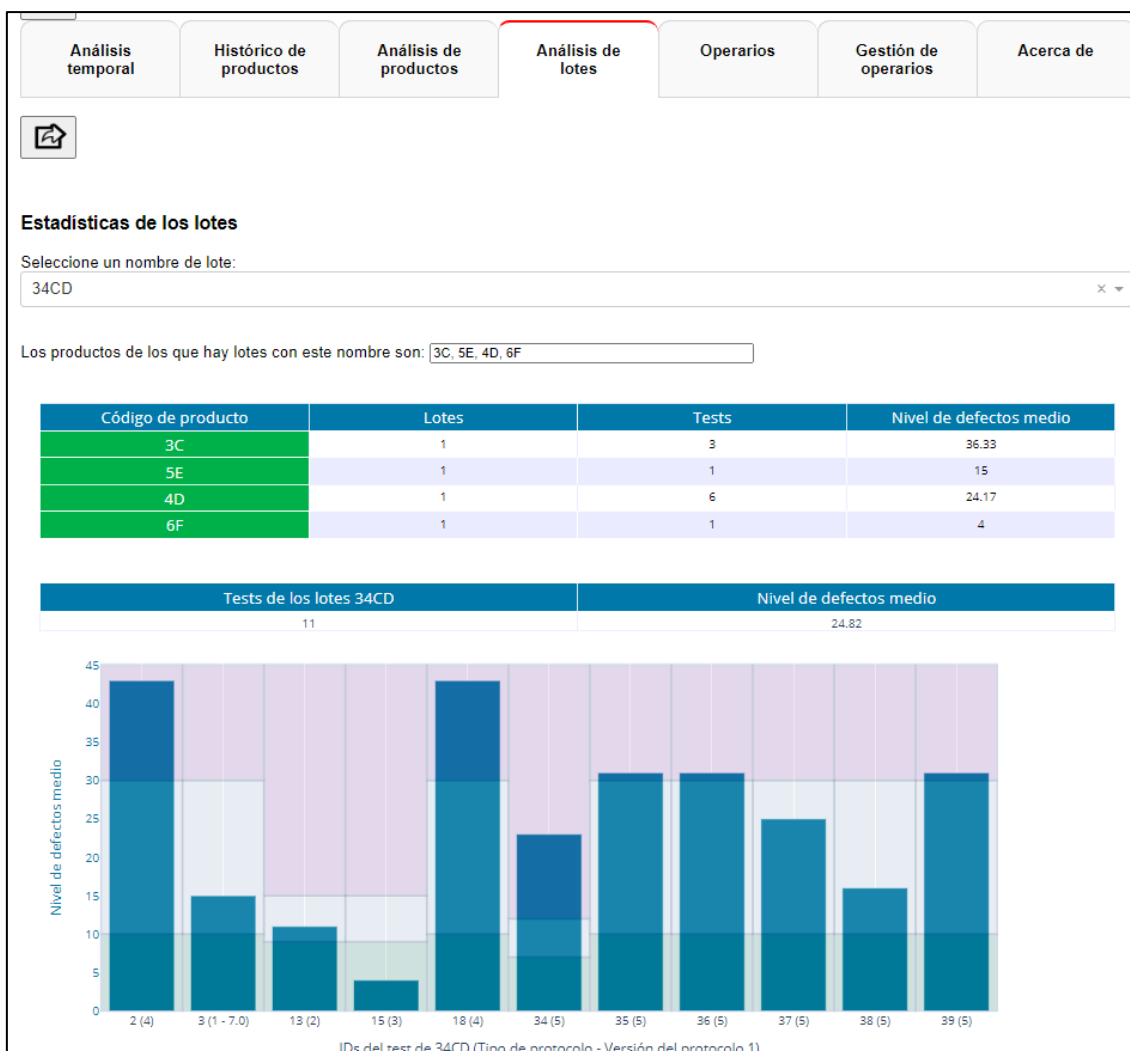


Figura 6 Vista de la pestaña Análisis de lotes

La pestaña “Operarios” muestra información acerca del número de tests supervisados por cada operario y el nivel de defectos medio asociado a los tests de cada uno de los operarios y en un rango de fechas elegido por el usuario.

También incluye información acerca de la fecha y hora a la que un operario supervisó un test determinado (Figuras Figura 7 y Figura 8).

3. DEMOSTRACIÓN DEL CUADRO DE MANDO DESARROLLADO

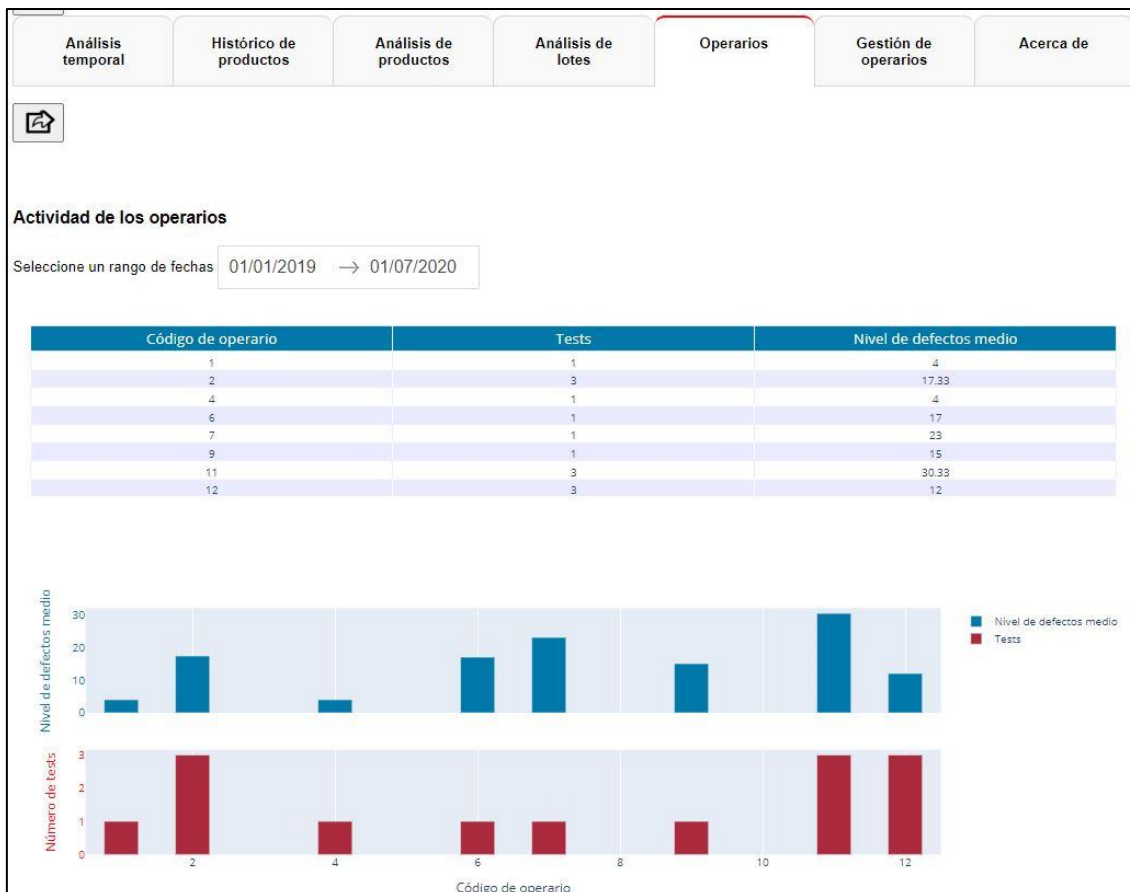


Figura 7 Número de tests y nivel de defectos medio para cada operario

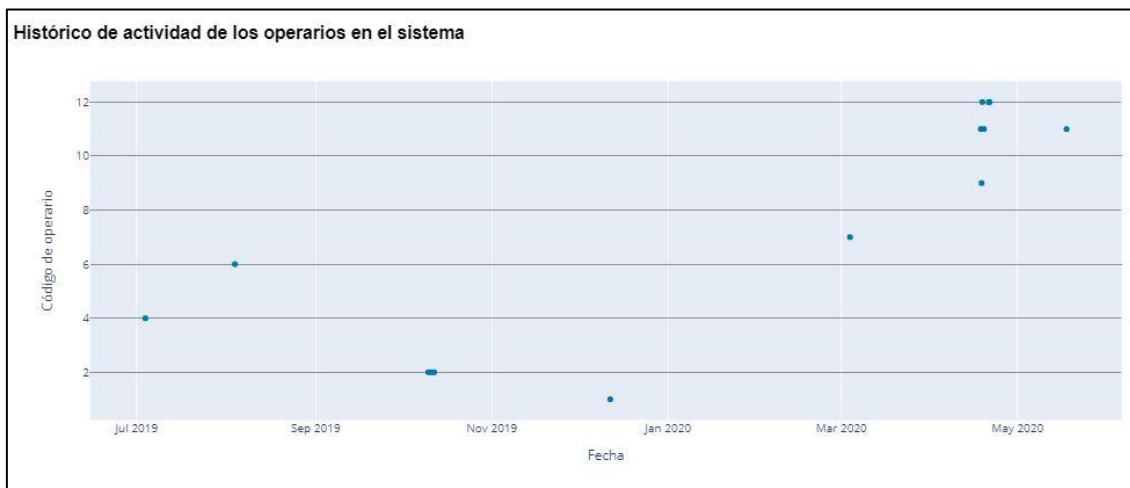


Figura 8 Cada punto representa un test realizado por un operario en un momento concreto. Si se amplía el eje horizontal lo suficiente, puede verse la fecha y la hora de cada test.

La pestaña “Gestión de operarios” permite listar los operarios en la empresa (Figura 9), añadir nuevos operarios, incluso si tienen el mismo nombre y apellidos que otros ya registrados, ya que se entiende que se puede dar esta circunstancia (Figura 10) y cambiar

3. DEMOSTRACIÓN DEL CUADRO DE MANDO DESARROLLADO

su estado entre “ACTIVO” e “INACTIVO” (Figuras Figura 11 y Figura 12). Basta con que el usuario seleccione una opción de las tres.

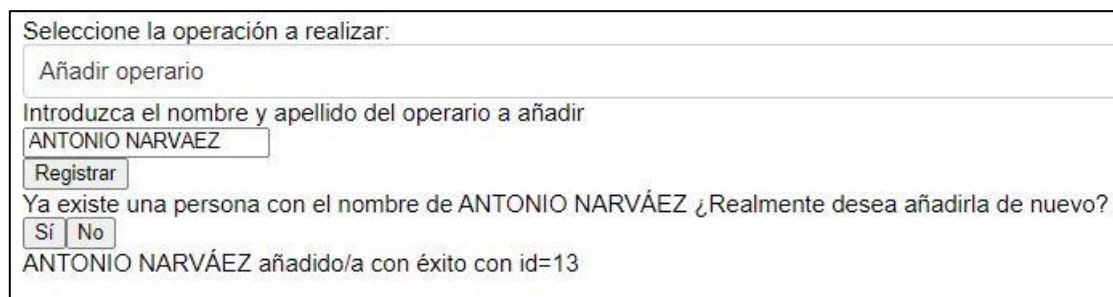


Seleccione la operación a realizar:

Listar operarios

Identificador	Nombre y apellidos	Estado actual
1	ANTONIO NÁRVAEZ	ACTIVO
2	EUGENIO BIGAS	ACTIVO
3	LETICIA SANTOS	ACTIVO
4	ALBERTO MARTÍN	INACTIVO
5	CARMEN HERNÁNDEZ	ACTIVO
6	LORENA FINCIAS	ACTIVO
7	MARÍA QUEVEDO	ACTIVO
8	ÁNGEL DOLORES	ACTIVO
9	ISABEL MUÑOZ	ACTIVO
10	CRISTINA ANTÚNEZ	ACTIVO
11	JOSE MULAS	INACTIVO
12	ENRIQUE BAJO	ACTIVO

Figura 9 Listar operarios. Se indica el identificador, el nombre y su estado en ese momento.



Seleccione la operación a realizar:

Añadir operario

Introduzca el nombre y apellido del operario a añadir

ANTONIO NARVAEZ

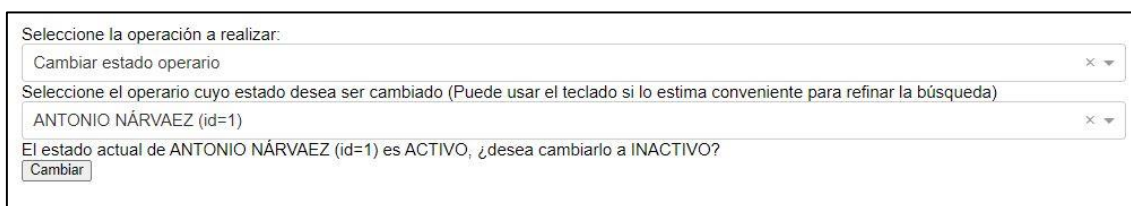
Registrar

Ya existe una persona con el nombre de ANTONIO NARVÁEZ ¿Realmente desea añadirla de nuevo?

Sí No

ANTONIO NARVÁEZ añadido/a con éxito con id=13

Figura 10 Añadir operario. Si ya existe ese nombre, pedirá confirmación para añadirlo.



Seleccione la operación a realizar:

Cambiar estado operario

Seleccione el operario cuyo estado desea ser cambiado (Puede usar el teclado si lo estima conveniente para refinar la búsqueda)

ANTONIO NÁRVAEZ (id=1)

El estado actual de ANTONIO NÁRVAEZ (id=1) es ACTIVO, ¿desea cambiarlo a INACTIVO?

Cambiar

Figura 11 Cambiar estado de un operario. Para ello, buscarle y pulsar el botón “Cambiar”

3. DEMOSTRACIÓN DEL CUADRO DE MANDO DESARROLLADO

Seleccione la operación a realizar:

Cambiar estado operario

Seleccione el operario cuyo estado desea ser cambiado (Puede usar el teclado si lo estima conveniente para refinar la búsqueda)

ANTONIO NÁRVAEZ (id=1)

El estado actual de ANTONIO NÁRVAEZ (id=1) es INACTIVO, ¿desea cambiarlo a ACTIVO?

Cambiar

Actualizado con éxito el estado de ANTONIO NÁRVAEZ (id=1) a INACTIVO

Figura 12 Cambiar estado de un operario. Puede volver a cambiarse el estado de un mismo operario las veces seguidas que se quiera.

La pestaña “Acerca de”, muestra información acerca de la aplicación (Figura 13)

Análisis temporal Histórico de productos Análisis de productos Análisis de lotes Operarios Gestión de operarios **Acerca de**

ACME - Sistema de análisis de niveles de calidad

Desarrollado por:

GCO - Universidad de Valladolid (España)

Desarrolladores técnicos: Pablo Marina & Illán González

Director del proyecto técnico: Ignacio de Miguel (ignacio.miguel@tel.uva.es)

Versión 1.0.0 – 1 de Julio de 2020

Figura 13 Vista de la pestaña Acerca de

A continuación, se expone cómo exportar la información contenida en cualquiera de las cinco primeras pestañas para crear un informe con un diseño elegante y parecido al de la aplicación.

Una vez el usuario esté en una de las pestañas citadas anteriormente, verá justo debajo del menú de pestañas a la izquierda un botón de “Exportar datos”. Cuando el usuario haya obtenido la vista deseada de todas las gráficas y tablas contenidas en esa pestaña, deberá de pulsar este botón.

Después de haberlo pulsado, aparecerá un mensaje indicándole que espere mientras se genera el documento en el servidor.

Cuando se haya terminado de generar el documento en el servidor, el mensaje anterior desaparecerá y, en su lugar, aparecerán dos botones nuevos de “Imprimir y “Descargar” con los aspectos de la Figura 14

3. DEMOSTRACIÓN DEL CUADRO DE MANDO DESARROLLADO



Figura 14 Botones de "Exportar", "Imprimir" y "Descargar"

Si el usuario pulsa el botón de imprimir, aparecerá un cuadro de impresión como el de la Figura 15, en el que podrá seleccionar la impresora a la que desea enviar el documento y las opciones de formato asociadas o si, por el contrario, desea salvar el documento en pdf. Cuando el usuario pulse "Aceptar", aparecerá un cuadro de vista previa antes de la impresión definitiva, la cual se iniciaría al pulsar en "Iniciar Impresión" (Figura 16).

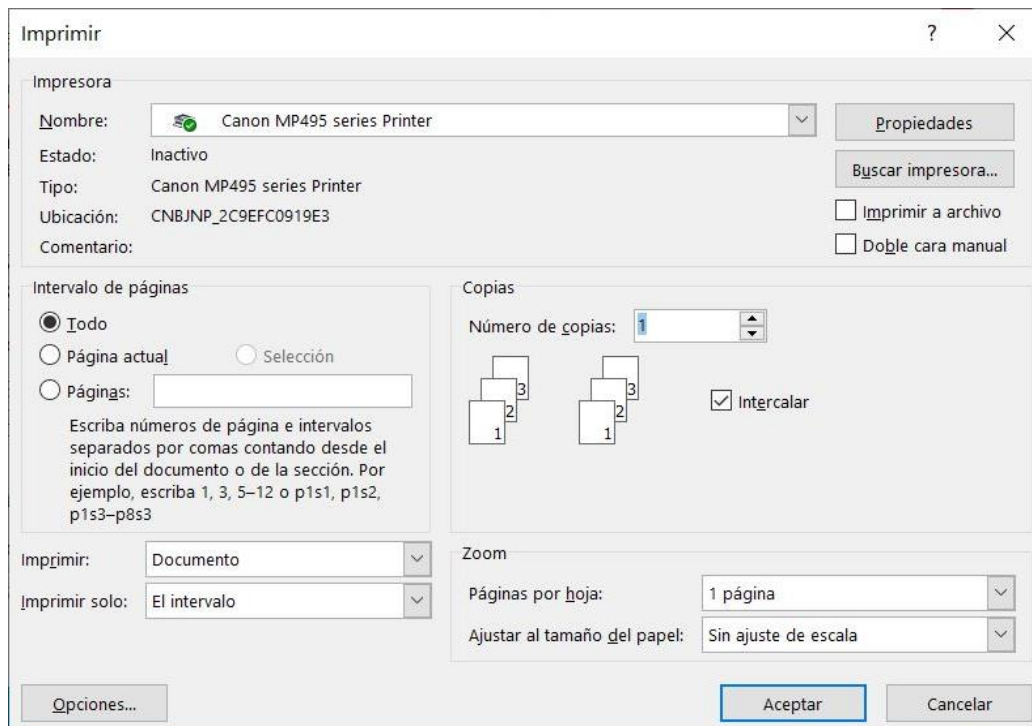


Figura 15 Cuadro de impresión del dashboard

3. DEMOSTRACIÓN DEL CUADRO DE MANDO DESARROLLADO

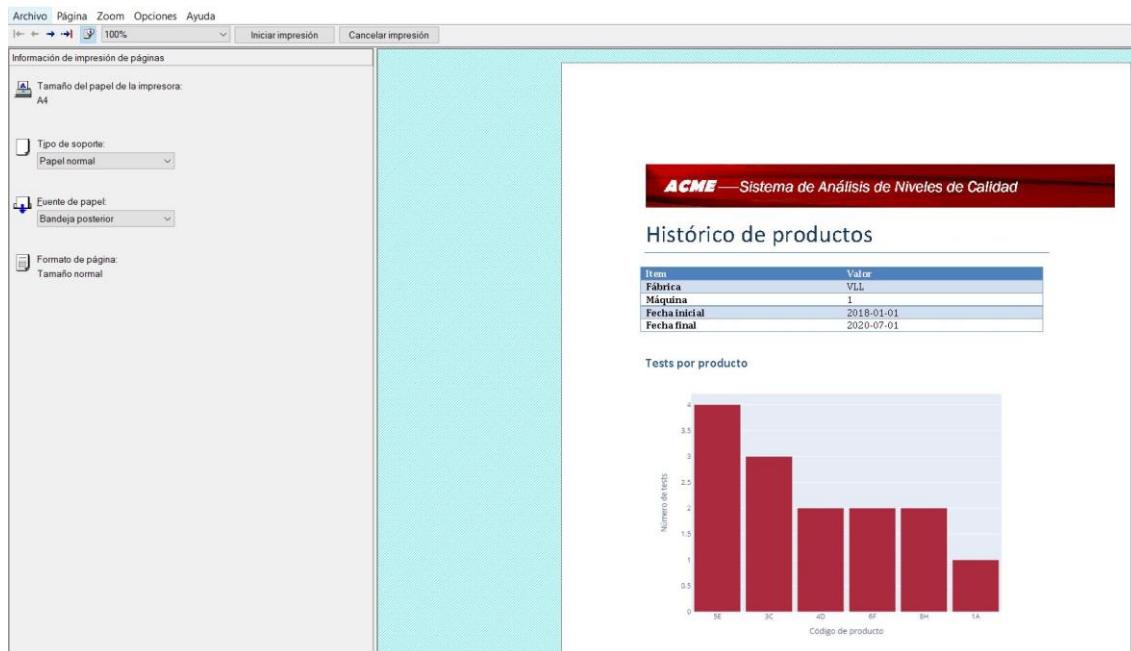


Figura 16 Vista previa antes de imprimir

De la misma forma, si el usuario pulsa el botón de descargar documento, obtendrá el informe correspondiente en formato *docx* de modo que podrá editarlo si lo considera conveniente.

3. DEMOSTRACIÓN DEL CUADRO DE MANDO DESARROLLADO

4. HERRAMIENTAS UTILIZADAS Y DECISIONES TOMADAS AL RESPECTO

Durante el proyecto, ha habido que hacer uso de varios tipos distintos de herramientas para llevar a cabo el cumplimiento de los objetivos planteados, desde el desarrollo del código hasta el despliegue de la aplicación. En algunos casos nos hemos encontrado con varias soluciones posibles para un mismo tipo de recurso, y hemos tenido que decantarnos por alguna. A continuación, se exponen todos los programas y aplicaciones que se han utilizado para llevar a cabo el desarrollo del cuadro de mando, así como las comparativas y decisiones tomadas entre las alternativas disponibles para una misma clase de herramienta.

4.1. ANACONDA

Ya que vamos a escribir y modificar código en *Python* para nuestra aplicación, necesitamos un entorno de desarrollo que lo albergue.

Anaconda es una distribución libre y de código abierto (*open-source*) para los lenguajes de programación *Python* y *R*.

Incluye varios módulos y librerías relacionados con estos lenguajes, gestionados a través del sistema *Conda*. En este proyecto se ha usado especialmente el Entorno de Desarrollo Integrado (IDE, *Integrated Development Environment*) *Spyder*.

Incluye también un terminal de comandos desde el cual se han instalado las librerías adicionales necesarias para llevar a cabo el proyecto y se han inicializado los distintos programas en código (Anaconda, 2020).

4.2. FLASK

Un marco de trabajo o *framework*, también es fundamental para contener el *layout* de la aplicación y su uso es un prerequisite para el empleo de la librería *dash*, la cual ha sido la herramienta clave para el desarrollo del cuadro de mando.

Flask es un marco de trabajo (*framework*) minimalista que actúa como contenedor para albergar aplicaciones web mediante el uso de URLs

Aparte del *framework*, incluye un servidor para servir la aplicación en una dirección IP y puerto determinados y que refresca la página de la aplicación automáticamente cuando se hace un cambio en el código sin tener que reiniciarlo, lo cual resulta de gran utilidad a la hora de depurar, aunque debido a su baja escalabilidad, dicho servidor se usará solo en máquina local para testear la aplicación, por lo que se requiere el uso de otro servidor que permita el despliegue (Flask, 2010).

4. HERRAMIENTAS UTILIZADAS Y DECISIONES TOMADAS AL RESPECTO

4.3. DASH

Para poder llevar a cabo la implementación de las distintas gráficas y tablas incluidas en nuestra aplicación, se necesita una librería dedicada a aplicaciones analíticas que implemente estos elementos visuales.

Una alternativa es *Bokeh*, aunque el GCO de la UVa, que es el grupo de investigación en el que se ha desarrollado este TFM, ya se había decantado en proyectos anteriores por la herramienta de código abierto *Dash*, disponible para *Python* y desarrollada por la compañía canadiense *Plotly* (Bokeh, 2019; Plotly, 2020).

Está implementada sobre *Flask* y cuenta con una alta capacidad de personalización de la interfaz de usuario. Ya que *dash* se visualiza a través de un navegador web, esto implica que es multiplataforma (*cross-platform*) y puede ser visualizado desde distintos tipos de dispositivos (portátiles, tabletas electrónicas, teléfonos móviles...) (*Plotly*, 2020).

4.4. SERVIDOR PARA DESPLEGAR LA APLICACIÓN

Debido a que el servidor que incorpora el marco de trabajo *Flask*, no escala bien a la hora de desplegar la aplicación para ser usada por varios usuarios, es necesario complementar el *framework* de *Flask* con algún servidor que complemente este déficit de escalamiento y que sea *standalone*, esto es, que sea independiente del marco de trabajo que se utilice. Por contrapartida, a diferencia de *Flask*, estos servidores necesitan ser reiniciados si se cambia código de la aplicación y suelen tardar unos segundos más en arrancar, por lo que se desaconseja su uso para testear y depurar la aplicación.

4.4.1. EL CICLO DE DESARROLLO DE SOFTWARE Y EL MODELO DTAP

En el ámbito de desarrollo de *software*, cada funcionalidad o caso de uso que es añadido a una aplicación determinada ha de pasar a través de una serie de procesos que corresponden al famoso Ciclo de Desarrollo de Software (SDLC, *Software Development Cycle*) (Figura 17). Estos procesos aseguran que el programa en el que se está trabajando tiene una hoja de ruta sólida y definida de principio a fin (Raphael-Rene, 2018; Rather & Bhatnagar, 2016).

4. HERRAMIENTAS UTILIZADAS Y DECISIONES TOMADAS AL RESPECTO

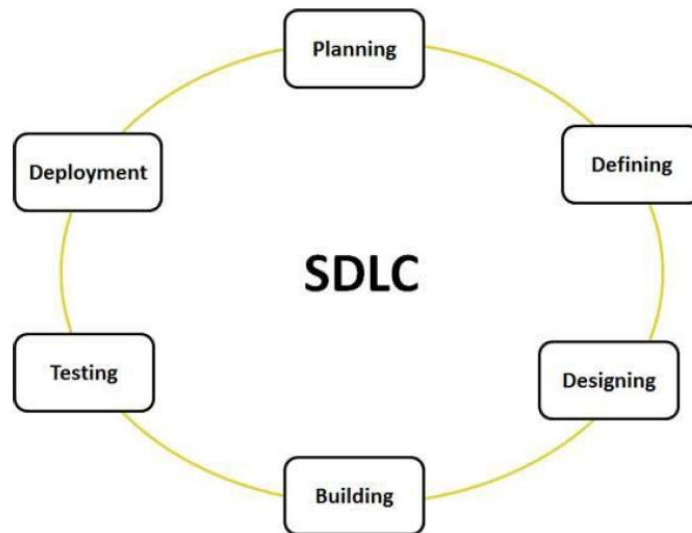


Figura 17 Ciclo SDLC (Rather & Bhatnagar, 2016)

El modelo DTAP (Desarrollo, Testeo, Aceptación y Producción, *Development, Testing, Acceptance and Production*) propone una aproximación para el testeo y el despliegue de *software* como parte del SDLC. Se compone de una serie de pasos que no han de entenderse como algo secuencial, sino como una serie de pasos que se ejecutan de forma paralela a lo largo de distintas iteraciones con algún tipo de metodología ágil como, por ejemplo, *Agile* (Figura 18). Esto permite flexibilizar el proceso y poder retroceder a etapas anteriores si las circunstancias así lo requieren (Offshore Software Solutions, 2019; Raphael-Rene, 2018).

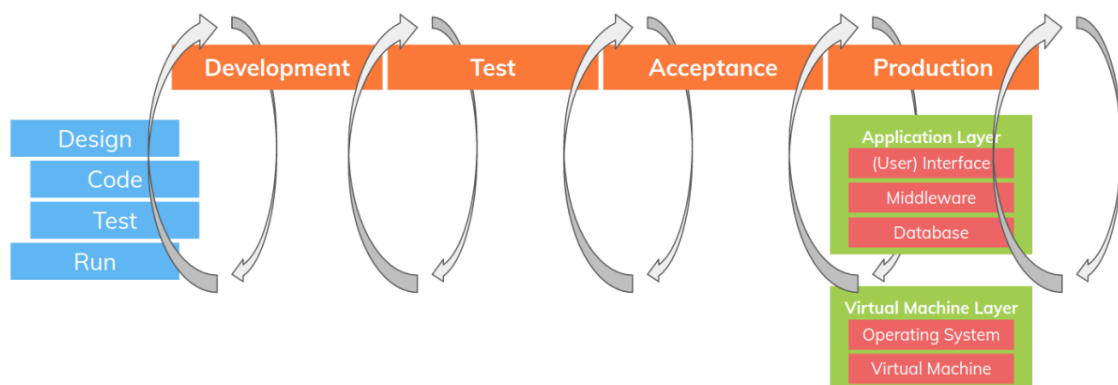


Figura 18 DTAP combinado con Agile (Offshore Software Solutions, 2019)

Cada fase del DTAP lleva implícita una forma de trabajar con el código hasta que la aplicación está finalizada y ve la luz.

4. HERRAMIENTAS UTILIZADAS Y DECISIONES TOMADAS AL RESPECTO

Se exponen a continuación las fases del DTAP (Raphael-Rene, 2018).

- Desarrollo
Es la etapa inicial del trabajo. En ella, los desarrolladores y programadores usan un servidor de desarrollo para implementar y testear distintas piezas de código por separado, que posteriormente serán incluidas en la aplicación
- Testeo
En esta nueva fase, el trabajo desarrollado anteriormente se despliega en un nuevo servidor de testeo. Esto permite que la aplicación sea accedida por el equipo de desarrollo para estresarla para verificar su correcto funcionamiento. Si se detecta alguna anomalía, se notifica a los desarrolladores para que la solucionen.
- Aceptación
Una vez que el código ha superado con éxito la etapa de testeo, este se introduce en un servidor de aceptación para intentar obtener el visto bueno del cliente, que deberá comprobar que la aplicación funciona de acuerdo a las especificaciones estipuladas en un principio. De no ser así, este puede avisar al equipo para someter el código a revisión.
- Producción
En este paso final, la aplicación ya ha obtenido todos los vistos buenos de quienes la han usado y testeado. Por lo que a la aplicación se le da luz verde para ser desplegada en un servidor de producción para su uso masivo.

Como podemos observar, se utilizan distintos tipos de servidores según la fase en la que nos encontremos del modelo. No obstante, esta aproximación es orientativa y no debe entenderse como algo cerrado e inflexible. Puede que el servidor de desarrollo sea el mismo que el de testeo y, en proyectos pequeños con un equipo muy reducido o que trabaja una sola persona, la fase de testeo se puede fusionar con la de aceptación.

4.4.2. SERVIDORES WSGI

Un servidor WSGI (*Web Server Gateway Interface*), implementa el lado de un servidor web de una interfaz WSGI para ejecutar programas en *Python*, mediante la invocación de objetos del lado de la aplicación (Figura 19) (Makai, 2019).

Tradicionalmente, un servidor web tenía una forma única de comunicarse con aplicaciones web escritas en *Python*. Esto implicaba que si se deseaba cambiar de servidor web, por cualquier motivo, ello conllevaba un rediseño de todo el código por parte de los desarrolladores, una tarea bastante tediosa y que suponía un gasto extra de tiempo y dinero (Alonso, 2019; Makai, 2019).

Con tal de poner solución a este problema, la comunidad de *Python* creó el estándar WSGI, recogido en la especificación PEP 3333 (*Python Enhancement Proposal*). El estándar WSGI, define una interfaz de bajo nivel y que establece entre el servidor web y el *framework* que alberga el programa una serie de reglas comunes para que se

4. HERRAMIENTAS UTILIZADAS Y DECISIONES TOMADAS AL RESPECTO

comuniquen entre sí, sea cual sea la implementación de ambos componentes. Esta separación se ilustra en la Figura 20 (Alonso, 2019).

De esta forma, la ventaja conseguida parece clara. El hecho de que WSGI sea agnóstico respecto del *framework* utilizado significa que podemos migrar nuestra aplicación a otro servidor web, sin que ello suponga un refactorizado del código. Y no solo eso. Otro punto fuerte es la gran escalabilidad que aportan los servidores WSGI. Pueden manejar miles de peticiones que provengan del servidor web y decidir en que forma comunicárselas a la aplicación, lo que implica que este tipo de servidores sean muy usados en la fase de producción, frente a otros servidores que carecen de la misma capacidad de escalar recursos, y que suelen ser usados en fase de desarrollo (Makai, 2019; Williams et al., 2019).

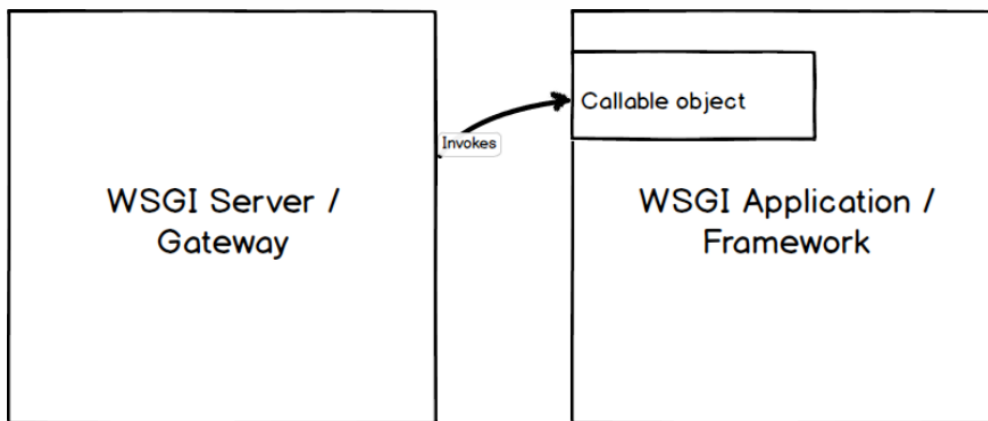


Figura 19 Un servidor WSGI invoca a objetos llamables en la aplicación escrita en Python (Makai, 2019)

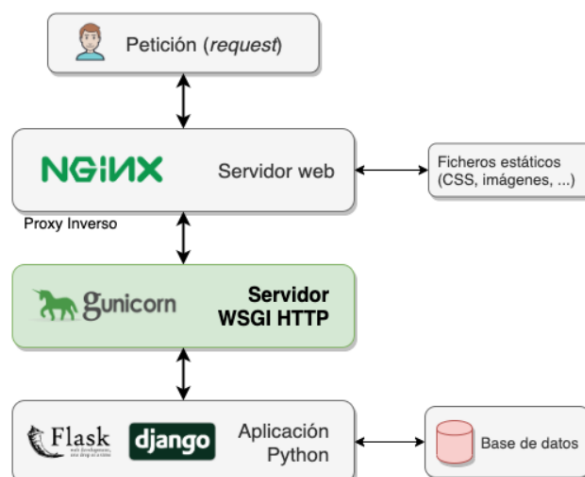


Figura 20 El estándar WSGI propone unas reglas de comunicación comunes entre el servidor web y el framework de la aplicación, sean cual sean estos (Alonso, 2019)

Entre los servidores WSGI de producción más populares destacan nombres como *Gunicorn*, *Waitress*, *CherryPy* o *Twisted*.

4. HERRAMIENTAS UTILIZADAS Y DECISIONES TOMADAS AL RESPECTO

Para escoger la mejor opción hay que barajar aspectos tales como compatibilidad, curva de aprendizaje, capacidad para soportar hilos (*threading*) o el año de lanzamiento de la primera versión.

Interesa utilizar un servidor WSGI de producción que sea compatible con *Windows*, que el sistema operativo en el que se desarrolla la aplicación y que sea sencillo de implementar.

Se realiza un estudio de las posibles alternativas.

- *Gunicorn*

Pensado para *UNIX*, *Gunicorn* es compatible con muchos tipos de *frameworks* distintos, incluyendo *Flask*. Inició su andadura en el 2010 y la versión 19.0, lanzada en 2014, ya incluye la funcionalidad de trabajar con múltiples hilos o *threads*.

Gunicorn tiene un sistema de configuración muy versátil a través de variables de entorno, ficheros de configuración o la propia línea de comandos, siendo esta última la más prioritaria, por lo que se podría almacenar el comando con la dirección IP, el puerto y el número de *threads* deseado en un *script* para ser ejecutado las veces que se desee (Chesneau, 2020).

- *Waitress*

Waitress es un servidor WSGI de producción puro, en otras palabras, no incorpora funcionalidades de *framework*, igual que *Gunicorn*, pero a diferencia de este, funciona, ya sea en *Windows*, *UNIX* o *MAC OS*.

Lanzó su primera versión, la 0.1, en el año 2011, y prácticamente desde el comienzo ya incluyó la posibilidad de realizar *multithreading*.

Este servidor destaca por su facilidad de uso, al estar compuesta su API por tan solo una función, que se añade al código en una sola línea y donde se especifica la dirección IP, el puerto y el número de *threads* deseado (Pylons Project Developers, 2020).

- *Cheroot*

Cheroot es el servidor HTTP para *Python*, perteneciente al *framework* *CherryPy*. Funciona sobre cualquier sistema operativo.

CherryPy es uno de los *frameworks* más sofisticados para *Python*, a la par que de los más antiguos. Su primera versión data del año 2002. Cuenta con una API muy amplia y sofisticada con diversas dependencias y clases entre las que se encuentran las de *threading*. Cuenta con un sistema de configuración muy avanzado a la par que complejo de usar mediante ficheros y diccionarios y a nivel

4. HERRAMIENTAS UTILIZADAS Y DECISIONES TOMADAS AL RESPECTO

global, de aplicación y de petición. Al principio, la funcionalidad de servidor WSGI estaba incluida dentro de una librería de *CherryPy*, aunque a partir de la versión 9.0 del año 2017, esta se ha movido de forma independiente a *Cheroot*.

Afortunadamente, este módulo independiente permite una configuración rápida y sencilla dentro del mismo código. Solo hay que especificar la dirección IP, el puerto, el número de *threads* deseado y ponerlo en marcha en apenas un par de líneas de código (CherryPy, 2020; CherryPy Team, 2020).

- *Twisted Web*

El *framework* Twisted basado en eventos es igualmente compatible con cualquier sistema operativo. Comenzó su andadura en el año 2002.

Está estructurado en varios módulos para desarrollar aplicaciones en Internet. El que nos interesa a nosotros es *Twisted Web*, iniciado en 2008, que incluye la posibilidad de poner en marcha un sitio web en un servidor WSGI en un puerto, dirección IP y número de *threads* mediante el concepto de “reactores”, que no son más que los eventos en bucle que permiten que corran dichas aplicaciones.

El único inconveniente de *Twisted Web*, es que sigue un proceso complejo de despliegue de la aplicación a través de distintas clases localizadas en distintas subdependencias que hay que ir importando, y puede resultar algo complejo para el desarrollador no experimentado (Twisted Matrix Labs, 2020b, 2020a).

El resumen de las características de cada alternativa se recoge en la Tabla 1

Tabla 1 Comparativa entre servidores WSGI de producción

	<i>Gunicorn</i>	<i>Waitress</i>	<i>Cheroot</i>	<i>Twisted Web</i>
Compatibilidad	UNIX	UNIX, Windows o MAC OS	UNIX, Windows o MAC OS	UNIX, Windows o MAC OS
Curva de aprendizaje	Muy baja	Muy baja	Baja	Alta
Año de lanzamiento	2010	2011	2017	2008
Threading	Sí	Sí	Sí	Sí

Tal y como se expuso anteriormente, necesitamos que la solución sea compatible con *Windows*, por lo que se descarta *Gunicorn*. También interesaba que la solución elegida tuviera una fácil implementación, por lo que *Twisted Web*, queda también excluido. Entre *Waitress* y *Cheroot*, es verdad que ambas ofrecen soluciones compatibles con *Windows* y sencillas de implementar. No obstante, se seleccionó *Waitress* por llevar más tiempo desarrollándose y porque tiene un mayor diferencial de simplicidad, respecto de *Cheroot*.

4. HERRAMIENTAS UTILIZADAS Y DECISIONES TOMADAS AL RESPECTO

4.5. MYSQL WORKBENCH

Para albergar la base de datos, sobre la cual se harán las distintas consultas para acceder a información contenida en ella y mostrarla en el cuadro de mando, o para modificarla, se utilizará *MySQL Workbench*.

Incluye una funcionalidad muy útil con la que se han importado las tablas con datos desde una hoja de cálculo *Excel* directamente a la base de datos en el período de pruebas, antes de trabajar con ficheros *.sql*.

Por otro lado, incorpora una interfaz de línea de comandos, o *shell* que se ha utilizado para realizar consultas en crudo a la base de datos y depurar las sentencias de código del programa relativas a consultas a la base de datos (*MySQL*, 2020).

Posteriormente, desde el código de la aplicación se han configurado los parámetros necesarios para realizar con éxito la conexión a la base de datos y obtener los datos pedidos por el cliente a través de la interfaz gráfica de la aplicación.

4.6. APACHE

Nuestro objetivo ahora es montar la aplicación en un directorio raíz fijo y accesible para todo el mundo. Para ello, utilizaremos el servidor de código abierto *Apache*, pensado para almacenar servicios del tipo Protocolo de Transferencia de Hipertexto (HTTP, *Hypertext Transfer Protocol*). Está estructurado mediante una sección *core* y luego incluye varios módulos en los cuales se incluyen gran parte de las funcionalidades básicas.

Dado que *Apache* no puede soportar directamente el layout de nuestra aplicación, tendremos que embeberla dentro de un *iframe* en un documento *html*, es decir, una ventana que conecta con el puerto donde se sirve la aplicación y, a través de la cual, se puede visualizar la interfaz de la misma e interactuar con ella.

El uso de este sistema aporta dos grandes ventajas. Para empezar, el puerto donde se sirve la aplicación puede cambiar, por razones de mantenimiento, pero el puerto donde funciona el servidor *Apache* es mucho más estable en el tiempo y es siempre el mismo, a no ser que se cambie en la configuración, lo que hace el acceso por parte del cliente mucho más sencillo. Después, proporciona un plus de privacidad, ya que podemos configurar una URL fija al documento *html*, que oculta los parámetros de ID de usuario que se mandan entre páginas mediante el parámetro GET, y que serían visibles en la URL sin este sistema.

Apache se configura a través de sencillos ficheros de texto plano, donde se incluyen las directivas necesarias para su funcionamiento como, por ejemplo, en qué puerto queremos que escuche, los tipos de fichero que puede ejecutar o la dirección del sistema de ficheros tomada como raíz en la URL. Estas directivas pueden ser globales o específicas para cada directorio (*Apache HTTP Server Project*, 2020).

4. HERRAMIENTAS UTILIZADAS Y DECISIONES TOMADAS AL RESPECTO

4.7. PAQUETE DE APLICACIONES PARA EL SERVIDOR LOCAL

A pesar de contar con un servidor de producción que sirva la aplicación en una dirección IP y un puerto para que sea accesible desde la red interna, con la finalidad de simplificar el manejo de *Apache* a través de alguna interfaz para activar y pausar este servicio y para acceder a los ficheros de configuración fácilmente, se va a integrar nuestro servidor dentro de un paquete de aplicaciones para servidor local. Existen varias alternativas, siendo *WampServer* y *XAMPP Control Panel* dos de los más populares.

Para poder elegir el paquete más adecuado a nuestras necesidades se han de tener en cuenta algunos parámetros tales como facilidad en el manejo, plataformas que soportan, módulos que incluye y complejidad en la instalación.

En este caso, nos interesa encontrar un paquete que sea fácil de manejar y que nos permita ser ágiles en la labor de desarrollo y, ya que solo vamos a necesitar *Apache*¹, que sea flexible a la hora de decidir qué componentes instalar.

A continuación se expone la discusión entre ambas alternativas (Temok, 2019).

- *WampServer*

WampServer es un sistema de infraestructura de páginas web que funciona solo en Windows. Los módulos que incluye son únicamente *Apache*, *PhpMyAdmin* y *MySQL*, que se instalan a la vez en el sistema de manera indivisible. *WampServer* cuenta con una interfaz gráfica consiste en un menú desplegable en la barra de iconos (Figura 21) que permite acceder al directorio del servidor y a los ficheros de configuración de cada componente así como para activarlos o pararlos (Figura 22).



Figura 21 Menú principal de WampServer

¹ Nótese que aunque *XAMPP Control Panel* también da opción de instalar *MySQL*, ya contábamos con esta aplicación por separado. Sin embargo, puede contemplarse en un futuro configurar dicha base de datos de forma íntegra dentro del propio paquete de aplicaciones para el servidor local

4. HERRAMIENTAS UTILIZADAS Y DECISIONES TOMADAS AL RESPECTO

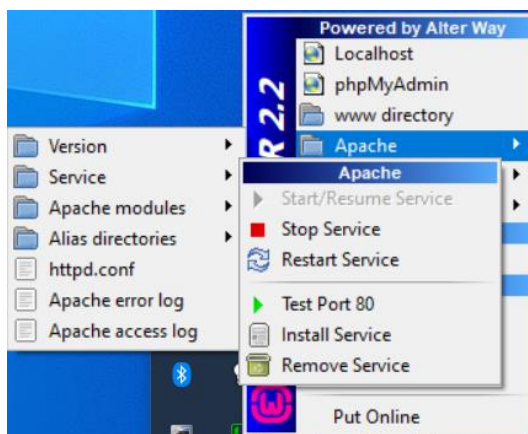


Figura 22 Acceso a la configuración y gestión de cada módulo

- XAMPP Control Panel
XAMPP Control Panel es otra solución de servidor local para páginas web, disponible para Windows, Linux y MAC OS y que incluye una gran variedad de módulos, aparte de los mencionados para *WampServer*. Otro de sus puntos fuertes es la posibilidad de seleccionar los paquetes que se desean durante el proceso de instalación (Figura 23). Presenta una interfaz muy intuitiva con una fila de botones para cada módulo que permite activarlos, detenerlos, acceder a su configuración y a sus logs, a golpe de vista (Figura 24).

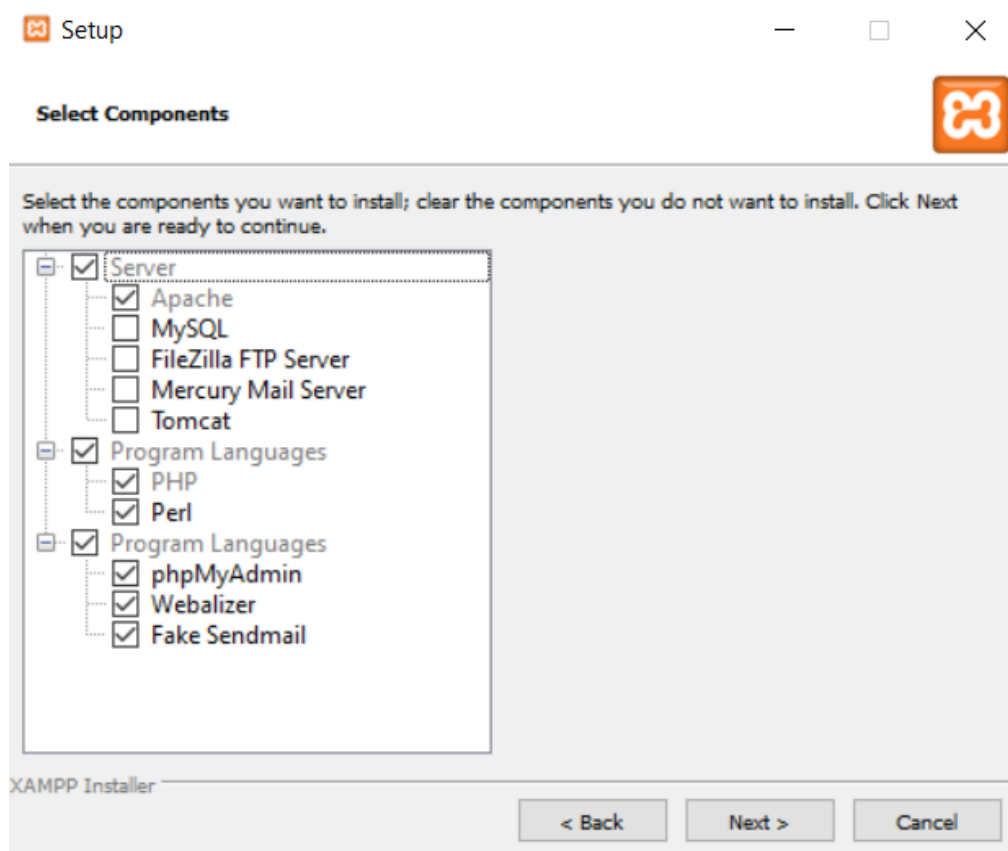


Figura 23 Selección de los componentes que se desean instalar en XAMPP Control Panel

4. HERRAMIENTAS UTILIZADAS Y DECISIONES TOMADAS AL RESPECTO

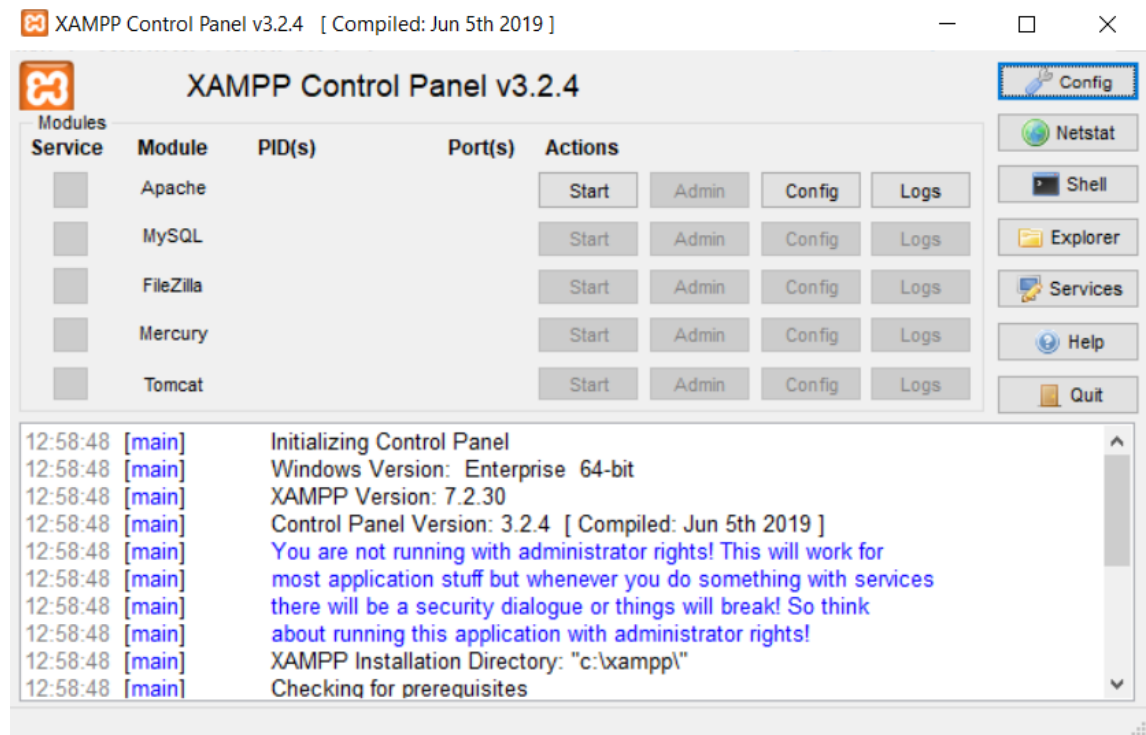


Figura 24 Interfaz gráfica de XAMPP Control Panel

La Tabla 2 resume las características de ambas aplicaciones.

Tabla 2 Resumen de las características de WampServer y XAMPP Control Panel

	<i>XAMPP Control Panel</i>	<i>WampServer</i>
<i>Manejo de la interfaz</i>	Fácil	Medio
<i>Sistemas operativos</i>	Windows, Linux, MAC OS	Solo Windows
<i>Cantidad de módulos disponibles</i>	Elevada	Baja (Solo <i>Apache</i> , <i>MySQL</i> y <i>PHPMysqlAdmin</i>)
<i>Instalación</i>	Avanzada (Selección de componentes)	Sencilla (Se instalan los tres módulos de forma inseparable)

Dado que estamos buscando una solución que permita flexibilizar los componentes que se instalan y sea intuitiva, nos decantaremos por *XAMPP Control Panel*. Es cierto que de momento solo necesitamos *Apache*. Sin embargo, si en un futuro necesitamos algún modulo adicional, siempre podremos reinstalar *XAMPP Control Panel* y añadir tan solo aquello que necesitemos (por ejemplo, incluir el sistema de gestión de bases de datos *MySQL* que tenemos funcionando por separado).

4. HERRAMIENTAS UTILIZADAS Y DECISIONES TOMADAS AL RESPECTO

5. CONSIDERACIONES REALIZADAS EN EL DISEÑO Y EL DESARROLLO DE LA APLICACIÓN

5. CONSIDERACIONES REALIZADAS EN EL DISEÑO Y EL DESARROLLO DE LA APLICACIÓN

A medida que se ha ido realizando este proyecto, ha habido que tomar algunas decisiones en lo que se refiere al diseño de la aplicación, con el fin de obtener el mayor rendimiento posible y brindar la mejor experiencia posible al cliente. Sirva este capítulo para exponer las más relevantes.

5.1. CONEXIÓN A LA BASE DE DATOS

Una de las aportaciones del TFM ha sido la lectura de datos desde una base de datos en lugar de hacerlo desde una hoja Excel. A la hora de permitir al código el acceso a la base de datos, lo que se ha hecho es utilizar la librería `pyodbc` de *Python* que acepta una cadena de texto que incluye la configuración necesaria para conectarse a la base de datos (nombre del servidor, uid, *password*...) y a su vez devuelve una variable que representa a una conexión a dicha base de datos y, cada vez que necesitemos acceder a la información contenida en ella, ya sea para obtener o modificar la información, asociamos una sentencia SQL a dicha conexión para que ejecute dicho comando sobre la base de datos.

De esta forma, si necesitamos mostrar cierta información por el cuadro de mando, en vez de importar un fichero Excel entero al principio del código, realizamos varias consultas que carguen solo ciertos datos en función del momento, aliviando de esta forma la carga computacional, sobre todo si la base de datos es muy voluminosa.

5.2. CREACIÓN Y EDICIÓN DE DATOS

Otra contribución importante de este TFM es la posibilidad de crear y modificar información en la base de datos. La aplicación implementa, a modo de ejemplo, una interfaz para añadir los nombres de los operarios que realizan los tests en la empresa, listar los ya existentes, y modificar su estado (activo/inactivo).

A la hora de dar de alta a un nuevo operario, se le pide al usuario que introduzca el nombre y apellido del mismo y, a través de un cursor que se obtiene de la variable de conexión mencionada en el apartado anterior, se define la consulta para insertar los datos del operario con el nombre introducido y cuyo estado por defecto será “ACTIVO”.

Por otro lado, si lo que queremos es cambiar el estado de un operario, el usuario puede realizar una búsqueda del operario cuyo estado desea ser cambiado de “ACTIVO” a “INACTIVO” o viceversa. Se vuelve a utilizar el mismo cursor empleado en la creación de un operario. Como se ha diseñado la base de datos para que pueda almacenar más de un operario con el mismo nombre, se especifica en todo momento el nombre con el identificador asociado al mismo (lo cual podría ser, por ejemplo, su número de empleado en la empresa), para eliminar las ambigüedades.

5. CONSIDERACIONES REALIZADAS EN EL DISEÑO Y EL DESARROLLO DE LA APLICACIÓN

5.3. EXPORTACIÓN DE DATOS A UN INFORME

Para poder exportar los datos que se muestran por pantalla, ya sea a un soporte físico (impresión en papel) o electrónico (por ejemplo, un informe en Word o una presentación en Power Point), se hace uso de librería `python-docx`. Cada vez que el usuario solicita la exportación de los datos que se están visualizando, esta librería se encarga de crear, del lado del servidor, un fichero *docx*, con una estructura similar al de la página web en cuestión que contenga el conjunto de gráficas y tablas mostradas en ese momento en el cuadro de mando, junto con los parámetros definidos por el usuario en el *dashboard*. Una vez creado este fichero, existen dos posibilidades: que se quiera descargar, del lado del cliente, ó que se quiera mandar a alguna impresora para plasmarlo sobre papel.

En el caso de descargar el fichero, se implementa un botón sobre la interfaz, con una dirección apuntando a ese fichero recién creado, de tal forma que el usuario puede obtener el documento en formato electrónico con solo un clic. Para ello se utiliza la función `send_from_directory` de *flask*.

Si, por el contrario, se desea imprimir el fichero, se usa la librería `win32com`, que implementa un cuadro de diálogo de impresión avanzada, que aparece en cuanto el usuario pulsa el botón de imprimir que se muestra en el propio cuadro de mando. Dentro del cuadro de diálogo, se puede elegir la impresora a la que enviar el fichero, si se quiere la impresión en blanco y negro o color, a doble cara, a 2 páginas por cara, etc. También es posible imprimir a otro fichero y salvarlo como pdf.

Se barajó la posibilidad de regenerar automáticamente el documento, cuando el usuario cambiaba el estado de algún control de la interfaz, haciendo que un código se ejecute utilizando ese cambio como *trigger*, pero este sistema daba lugar a inconsistencias, ya que el documento generado incluía los datos nuevos de los controles, pero las gráficas antiguas, ya que aún no les había dado tiempo a actualizarse. Una solución sería poner no los controles como *triggers*, sino las gráficas y tablas. Aún así podría haber mezcla de versiones de elementos visuales antiguos y nuevos en el informe si cambia más de una gráfica y/o tabla y siguen los *triggers* implementados en paralelo. A su vez, otra forma de solventar esto es poner los *triggers* en cascada, de tal forma que el cambio de una gráfica dispara el cambio de la siguiente, y el cambio de la última de todas dispara la generación del informe, esta vez sí, con todos los datos actualizados. No obstante, esto implicaría complicar y refactorizar demasiado el código, y se optó por dejar el botón de exportar.

Sin embargo, una limitación principal que se encuentra al implementar este botón de exportación, es que si el usuario ya le ha dado a exportar, y por tanto ha hecho aparecer los botones de imprimir y descargar, quedando de esta forma los tres a la vista, si el usuario cambia algo en los controles y ve que los botones de imprimir y descargar siguen a la vista, puede olvidar que tiene que darle de nuevo a exportar para que se genere dicho informe de nuevo y pulsar directamente imprimir o descargar y, por tanto, obtiene de nuevo la versión antigua. Ocurriría lo mismo si el usuario se acuerda de exportar primero, pero decide imprimir o descargar antes de que se actualice el documento. Este último problema se ha intentado solventar haciendo que aparezca un mensaje de espera tras

5. CONSIDERACIONES REALIZADAS EN EL DISEÑO Y EL DESARROLLO DE LA APLICACIÓN

pulsar el botón de exportar, y que desaparezca cuando los otros dos botones ya están disponibles. No obstante, esto solo se ha conseguido para la primera pulsación dentro de una misma pestaña con las dos opciones ocultas. Si están a la vista, no se ha conseguido que aparezca el mensaje. En este sentido, se propone en un futuro integrar el *trigger* de exportar, dentro de los botones de impresión y descarga, dejando estos siempre a la vista, y que se genere la nueva versión al pulsar cualquiera de los dos, como paso previo a la función principal de dicho botón. Cada vez que se esté generando el documento, para que el usuario vea que el programa ha recibido la orden, poner un mensaje de espera hasta que esté listo.

Otra restricción importante que se ha tenido en cuenta tiene que ver con la edición de un informe ya creado, cuando el usuario, habiendo exportado ya una vez, decide cambiar el estado de los controles y exporta de nuevo. Con esto, el fichero en el servidor se sobrescribe sin problemas, pero a la hora de descargarlo o imprimirlo, siempre tiene en cuenta la primera versión, aunque ya no exista. En este caso, la solución ha sido poner un número de versión al final del nombre de cada informe del lado del servidor. Cuando el usuario exporta de nuevo un informe en una misma sesión, la versión anterior se borra y se genera una nueva, aumentando dicho número en una unidad, dependiendo del número de veces que el botón de exportar sea pulsado en una misma pestaña durante la conexión. Ahora bien, del lado del cliente, el informe siempre se descargará con el nombre original sin número.

5.4. IDENTIFICADORES DE SESIÓN

Una de las principales dificultades que se ha tenido durante el proyecto, ha sido la de poder soportar múltiples usuarios sin que haya sobreescritura de los informes que se acaban de mencionar. Se ha optado por identificar a cada usuario conectado en esos momentos mediante un identificador de sesión, generado con la función `uuid4()`, de la librería `uuid`, que genera una cadena alfanumérica aleatoria de 16 bytes en caracteres hexadecimales, la cual es escondida en un elemento no visible del *layout*.

De esta forma, cuando un usuario desee exportar unos datos a un documento Word, este documento se genera y almacena en el servidor dentro de una carpeta que tiene asignada y que tiene por título esa cadena identificadora. Aunque es altamente improbable, se comprueba que dicho número de sesión no está cogido, pero si se da el extraño caso de que realmente ya está cogido, se genera otro.

Para garantizar una asignación correcta de distintos números de sesión, se ha estimado conveniente la implementación de una portada de la aplicación que contiene el logo de la aplicación y un botón para acceder a ella. Cuando el usuario pulsa, se asigna el identificador correspondiente. Cuando el usuario finaliza la sesión mediante un botón de *logout* que redirige de nuevo a la portada de la aplicación, la carpeta asociada a ese cliente se destruye, junto con todos los informes que ha exportado durante el tiempo de conexión y ese identificador queda liberado.

Se entiende que habrá usuarios que no utilizarán el botón de *logout* cuando terminen de utilizar el cuadro de mando y simplemente cerrarán el navegador, por lo que los

5. CONSIDERACIONES REALIZADAS EN EL DISEÑO Y EL DESARROLLO DE LA APLICACIÓN

documentos no se destruirán al término de la conexión. En este sentido, se ha optado por implementar un sistema que compruebe, al entrar un nuevo usuario, aquellas carpetas que lleven en el servidor más de un día sin que se creen o modifiquen nuevos documentos en su interior, asumiendo que ese usuario ya no está conectado o interesado en esos documentos, aunque en el extraño caso de que aún siguiera conectado, la carpeta se crearía de nuevo sin problemas, aunque vacía de nuevo.

En este apartado cabe destacar una limitación principal. Si el usuario refresca la página una vez conectado (por ejemplo, pulsa la tecla F5), se asigna un número de sesión distinto al primero y todos los documentos que se exporten, lo harán dentro de una nueva carpeta con ese nuevo identificador, y al finalizar sesión, solo se destruirá la carpeta con el número de identificación vigente. Los informes exportados en la carpeta antigua quedan inaccesibles, olvidados y dentro del servidor hasta que pase el tiempo suficiente para que la carpeta se elimine automáticamente, tal y como se ha descrito anteriormente. Se ha intentado usar la herramienta *sessions* de *flask* para almacenar el identificador de forma persistente y que no cambie si se refresca la página, aunque no se ha encontrado una manera factible de llevarlo a cabo.

5.5. PUESTA EN MARCHA DEL SERVIDOR

Para desplegar la aplicación y hacerla visible de cara al exterior, se ha utilizado la librería de *Python* correspondiente al servidor de producción que hemos elegido, en nuestro caso *waitress*. Mediante una única línea de código al final del mismo, especificamos la dirección IP privada de nuestro ordenador, el puerto donde queremos que se ejecute la aplicación y el número de *threads* deseado. El ordenador con el que se ha llevado a cabo el despliegue tiene 4 procesadores o *cores*. Si estimamos en torno a 3 *threads* por *core*, la aplicación puede funcionar bien con 12 *threads*, por lo que ese es el número de hilos con el que corre nuestro programa.

Más tarde, para embeber la aplicación, de tal forma que sea soportada por *Apache*, se ha escrito un documento en lenguaje html que incluye un marco o *iframe* a través del cual se visualiza la aplicación, y que debe tener configurados la misma dirección y puerto que hemos especificado en el código con *waitress*.

Después, hay que especificar en el fichero de configuración de *Apache*, la dirección URL raíz que se tomará como referencia a la hora de acceder al documento html en nuestro sistema de ficheros, la dirección IP del servidor local (la misma de antes) y el puerto en el que queremos que escuche *Apache* y que ha de ser distinto al que configuramos para conectar el documento html y la aplicación.

Por último, se arranca la aplicación en *Python* desde una consola de comandos y se activa el servicio *Apache* desde la interfaz de *XAMPP Control Panel* y, de esta forma, ya se podría acceder al documento html con la aplicación desde cualquier dispositivo que esté conectado a la misma red privada que el servidor.

6. CONCLUSIONES

6. CONCLUSIONES

6.1. CUMPLIMIENTO DE LOS OBJETIVOS

Al principio de este TFM se expusieron aquellos objetivos que se perseguían. La meta principal consistía en desarrollar un *dashboard* de una supuesta empresa que mostrase los KPIs más relevantes de sus procesos de testeo de productos a partir de una versión más simple desarrollada dentro del GCO de la UVa. Esta meta se ha conseguido. Además, en relación con los objetivos específicos:

- El primer objetivo trataba de leer la información necesaria contenida en una base de datos para representarla en el cuadro de mando mediante gráficas y tablas. Sí que se ha cumplido, pues la aplicación interacciona con la base de datos en lugar de con una hoja de cálculo.
- La siguiente meta era añadir y modificar la información almacenada en la base de datos desde el *dashboard*. Se ha logrado al implementar un sistema para añadir y cambiar el estado de los operarios de la empresa.
- El siguiente objetivo hablaba de exportar documentos con elementos visuales mediante alguna funcionalidad que lo permitiera. También se ha cumplido, al utilizar una librería que permite crear documentos en *Word* a los cuales añadir el contenido deseado.
- El siguiente cometido consistía en imprimir el contenido de distintas pestañas de la aplicación con un diseño más refinado que el que se obtendría dando directamente a imprimir en el navegador. Se ha logrado, implementar un cuadro de dialogo para imprimir el informe según las opciones elegidas (ej. a doble cara, en blanco y negro...)
- Por último, hemos encontrado una forma de desplegar nuestra aplicación en un servidor accesible desde la red privada dentro de la empresa.

6.2. PRINCIPALES IDEAS EXTRAÍDAS

- Un *dashboard* puede llegar a resultar una herramienta muy útil en el ámbito empresarial. A partir de unos objetivos bien definidos, se establecen los parámetros a monitorizar, cuyos valores determinarán las decisiones que se tomen y los cambios que se introduzcan en los procesos internos de fábrica. Además, pueden añadirse otros servicios como el de la exportación de datos en distintos formatos, implementado en este proyecto.
- No es aconsejable importar al código todos los datos (ej. mediante un fichero de *Excel*) si estos son muy numerosos. Resulta más práctico preparar una base de datos con la información y que el código se conecte con ella según necesidad para editar la misma, o cargar subconjuntos de datos en la aplicación, tal y como se ha hecho en este TFM.
- El modelo DTAP para desarrollo de *software* propone cuatro etapas y un tipo de servidor distinto por etapa, aunque no es algo estricto. En este proyecto, para

6. CONCLUSIONES

servir la aplicación, se han usado inicialmente solo servidores de testeo, para hacer pruebas menores y que refrescan la aplicación cada vez que se salva el código, y finalmente de producción, para comprobar el funcionamiento global de la aplicación y que son más escalables que los primeros.

- En ocasiones, a la hora de necesitar una herramienta concreta, ha habido que escoger entre varias opciones, de las que se ha elegido una. Esto no quiere decir, que la alternativa escogida sea mejor que las otras en cualquier contexto, si no que depende de las prioridades que se hayan establecido en cada caso.

6.3. LIMITACIONES Y LÍNEAS FUTURAS

El desarrollo del presente proyecto no ha estado exento de limitaciones y restricciones que pueden haber condicionado el aspecto y funcionamiento finales de nuestra aplicación:

- Queda poco intuitivo que haya a la vista a la vez un botón de generar informe junto con los de imprimir o descargar informe. Sin embargo, las funciones utilizadas impedían aunar las dos acciones para que se ejecutaran una a continuación de la otra. Por otro lado, se ha implementado un mensaje de espera para que apareciera después de que el botón de exportar fuera accionado, para indicar que la orden ha sido recibida y que, si los botones de imprimir y descargar están a la vista, se indique al usuario que espere a que la nueva versión del documento esté lista. Sin embargo, solo se ha logrado que aparezca la primera vez, cuando aún no hay botones de impresión y descarga. En futuros proyectos que sigan esta línea se puede reestructurar el código para dejar las opciones de impresión y descarga siempre a la vista y, cada vez que se seleccionen una de los dos, generar el informe en el acto mientras aparece el mensaje incitando a la espera.
- Como alternativa al apartado anterior, sería posible actualizar un informe en el servidor sin tener que esperar a que el usuario quiera imprimirlo o descargarlo, poniendo como *triggers* las gráficas y tablas de la interfaz implementados en cascada. De esta forma, la actualización de la última tabla o gráfica dispara la creación del informe listo para cuando el usuario quiera imprimirlo o descargarlo y sin datos incoherentes. En este caso, poner o no el mensaje de espera es de libre elección aunque se recomienda ponerlo si es una actualización pesada.
- A la hora de sobrescribir un informe ya generado antes, se ha visto que aunque en el servidor se actualiza sin problemas, el cliente obtiene siempre la versión antigua, lo que nos obliga a cambiar el nombre del fichero cada vez para evitar este problema. Se propone encontrar una forma para poder sobrescribir el fichero con el mismo nombre y solventar el problema de la primera versión.
- Si el usuario se ve obligado por cualquier causa a refrescar la página o si simplemente lo hace de forma inintencionada una vez ya se le ha asignado un identificador de sesión, el sistema le reconocerá como un nuevo usuario distinto y le asignará otro identificador distinto y el informe que hubiese exportado antes se perdería y tendría que exportarlo de nuevo. Se ha intentado utilizar la herramienta `sessions` de *flask*, para asignar el identificador a una sesión y hacerlo llegar al *layout*, aunque no se ha encontrado una forma factible de hacerlo.

6. CONCLUSIONES

Se sugiere en un futuro emplear esta herramienta para implementar sesiones persistentes que garanticen que los datos asociados a una sesión no se pierden si se refresca una página.

- En este proyecto, se ha desarrollado una funcionalidad para gestionar los operarios de la empresa. En trabajos posteriores, se podrían gestionar aspectos adicionales relacionados con los tests desde la interfaz de la aplicación.

6. CONCLUSIONES

7. BIBLIOGRAFÍA

7. BIBLIOGRAFÍA

- Aceto, G., Persico, V., Pescapé, A., & Member, S. (2019). A Survey on Information and Communication Technologies for Industry 4.0: State-of-the-Art, Taxonomies , Perspectives , and Challenges. *IEEE Communications Surveys & Tutorials*, 21(4), 3467–3501.
- Alcácer, V., & Cruz-Machado, V. (2019). Scanning the Industry 4.0: A Literature Review on Technologies for Manufacturing Systems. *Engineering Science and Technology, an International Journal*, 22(3), 899–919. <https://doi.org/10.1016/j.jestch.2019.01.006>
- Alonso, N. (2019). ¿Qué es un WSGI? - Medium. Retrieved May 28, 2020, from <https://medium.com/@nachoad/que-es-wsgi-be7359c6e001>
- Anaconda. (2020). Individual Edition | Anaconda. Retrieved May 5, 2020, from <https://www.anaconda.com/products/individual>
- Apache HTTP Server Project. (2020). Welcome! - The Apache HTTP Server Project. Retrieved April 28, 2020, from <http://httpd.apache.org/>
- Bokeh. (2019). Bokeh 2.1.1 Documentation. Retrieved June 24, 2020, from <https://docs.bokeh.org/en/latest/index.html>
- Bradea, I., Sabău-Popa, D. C., & Boloş, M. I. (2016). USING DASHBOARDS IN BUSINESS ANALYSIS. In *Annals of the University of Oradea. Economic Sciences. Tom XXIII* (pp. 851–856). Oradea & Bucharest, Romania.
- Cherrypy. (2020). Welcome to Cherroot documentation! — cherroot 8.3.1. Retrieved May 20, 2020, from <https://docs.cherrypy.org/projects/cherroot/en/latest/>
- CherryPy Team. (2020). CherryPy Documentation, 57–61, 99–115.
- Chesneau, B. (2020). Unicorn Documentation Release 20.0.4, 14, 73.
- Flask. (2010). Welcome to Flask — Flask Documentation (1.1.x). Retrieved April 27, 2020, from <https://flask.palletsprojects.com/en/1.1.x/>
- Makai, M. (2019). WSGI Servers - Full Stack Python. Retrieved May 28, 2020, from <https://www.fullstackpython.com/wsgi-servers.html>
- MySQL. (2020). MySQL. Retrieved April 27, 2020, from <https://www.mysql.com/>
- Offshore Software Solutions. (2019). Our Development : Offshore Software Solutions. Retrieved May 27, 2020, from <https://www.offshoresoftware.solutions/development/>
- Plotly. (2020). Introduction | Dash for Python Documentation | Plotly. Retrieved April 27, 2020, from <https://dash.plotly.com/introduction>
- Popkova, E. G., Ragulina, Y. V., & Bogoviz, A. V. (2018). *Industry 4.0: Industrial Revolution of the 21st century*. (J. Kacprzyk, Ed.), *Nature* (1st ed., Vol. 417). Cham, Switzerland: Springer. <https://doi.org/10.1038/nj6885-03a>
- Pylons Project Developers. (2020). Waitress Documentation - Release 1.4.3, 1, 10, 27–89.
- Raphael-Rene, D. (2018). Development, Testing, Acceptance, Production- DTAP. Retrieved May 27, 2020, from <https://www.gratasoftware.com/what-is-each-server-for-development->

7. BIBLIOGRAFÍA

test-uat-or-staging-demo-and-production/

Raptis, T. P., Passarella, A., & Conti, M. (2019). Data management in industry 4.0: State of the art and open challenges. *IEEE Access*, 7, 97052–97093. <https://doi.org/10.1109/ACCESS.2019.2929296>

Rather, M. A., & Bhatnagar, V. (2016). A comparative study of sdlc model. *International Journal of Application or Innovation in Engineering & Management*, 4(10), 24.

Shamsuzzoha, A., Hao, Y., Helo, P., & Khadem, K. (2014). Dashboard User Interface for Measuring Performance Metrics : Concept from Virtual Factory Approach. In *Proceedings of the 2014 International Conference on Industrial Engineering and Operations Management* (pp. 124–133). Bali, Indonesia.

Temok. (2019). A Comparative Study of WAMP vs XAMPP: Which is the Best Suitable Local Server for Web Development? | Temok Hosting Blog. Retrieved April 28, 2020, from <https://www.temok.com/blog/wamp-vs-xampp/>

Twisted Matrix Labs. (2020a). Twisted Documentation, 5, 405–477.

Twisted Matrix Labs. (2020b). Twisted Matrix API. Retrieved May 23, 2020, from <https://twistedmatrix.com/documents/current/api/moduleIndex.html>

Williams, M., Benfield, C., Warner, B., Zadka, M., Mitchell, D., Samuel, K., & Tardy, P. (2019). *Expert Twisted*. (W. Spahr, J. Gennick, J. Markham, & J. Balzano, Eds.). New York, NY: Apress.

Apéndice A. ABREVIATURAS Y ACRÓNIMOS

- BI Inteligencia de Negocio (*Business Intelligence*)
- DTAP Desarrollo, Testeo, Aceptación y Producción (*Development, Testing, Acceptance and Production*)
- GCO Grupo de Comunicaciones Ópticas
- HTTP Protocolo de Transferencia de Hipertexto (*Hypertext Transfer Protocol*)
- IDE Entorno de Desarrollo Integrado (*Integrated Development Environment*)
- KPI Indicador Clave de Rendimiento (*Key Performance Indicator*)
- PEP Propuesta de Mejora de *Python* (*Python Enhancement Proposal*)
- TFM Trabajo Fin de Máster
- SDLC Ciclo de Desarrollo de Software (*Software Development Lifecycle*)
- SQL Lenguaje de Consulta Estructurada (*Structured Query Language*)
- URL Localizador de Recursos Uniforme (*Unified Resource Location*)
- UVa Universidad de Valladolid
- WSGI Servidor de Puerta de Enlace de la Interfaz Web (*Web Server Gateway Interface*)

Apéndice A. ABREVIATURAS Y ACRÓNIMOS

Apéndice B. MATERIAL UTILIZADO

Los medios, clasificados como documentación, hardware y software, que han sido necesarios para la elaboración del presente TFM son los siguientes:

DOCUMENTACIÓN

Servicio de la Biblioteca de la Escuela Técnica Superior de Ingenieros de Telecomunicación.

Documentación oficial del módulo *dash* de Plotly®.

Documentación oficial de MySQL® Workbench.

Documentación oficial de HTTP Apache® Server Project.

Documentación oficial de CherryPy®.

Documentación oficial de Cheroot®.

Documentación oficial de Twisted® Web.

Documentación oficial de Waitress®.

Documentación oficial de Gunicorn®.

Documentación oficial de Flask®.

Resto de reportes científicos y libros disponibles vía web.

HARDWARE

Ordenador portátil con Intel® Core(TM) i7-6700HQ CPU @ 2.60 GHz 2.60 GHz, RAM de 8 GB y 4 *cores*.

Memoria USB Kingston de 32 GB, USB 3.1

CD-ROM.

SOFTWARE

Microsoft® Windows 10.

Microsoft Office Professional Plus 2013. Versión 15.0.4569.1506

Mendeley® Desktop. Versión 1.17.13

MySQL® Workbench. Versión 8.0.19

Apéndice B. MATERIAL UTILIZADO

Anaconda3[®]. Versión 2019.10

Google Chrome. Versión 83.0.4103.106

Spyder[®]. Versión 3.3.6

Microsoft[®] Teams. Versión 1.3.00.13565

Skype[®]. Versión 15.61.87.0

Adobe[®] Acrobat Reader. Versión 20.009.20067

XAMPP[®] Control Panel. Versión 7.2.31-0

Git Bash. Versión 2.25.0

HTTP Apache[®] Server Project. Versión 2.4.43

Cheroot[®]. Versión 8.3.0

Waitress[®]. Versión 1.4.3

Twisted[®] Web. Versión 20.3.0

Flask[®]. Versión 1.1.2

Dropbox[®]. Versión 99.4.501

Apéndice C. PRESUPUESTO

Ejecución material

Ordenador portátil con Intel® Core(TM) i7-6700HQ CPU @ 2.60 GHz 2.60 GHz, RAM de 8 GB y 4 *cores* (prorrateado).....111 €

Memoria USB Kingston de 32 GB, USB 3.1.....10 €

TOTAL EJECUCIÓN MATERIAL.....121 €

Material fungible

Gastos de impresión y papelería.....120 €

TOTAL MATERIAL FUNGIBLE.....120 €

Honorarios del proyecto

450 horas a 30 € la hora².....13.500 €

Subtotal del proyecto.....13.741 €

I.V.A. aplicable

21% del subtotal del proyecto.....2.885,61 €

TOTAL DEL PROYECTO.....16.626,61 €

El total del presupuesto asciende a:

DIECISEIS MIL SEISCIENTOS VEINTISEIS EUROS CON SESENTA Y UN CÉNTIMOS DE EURO

En Valladolid, julio de 2020

Fdo: Pablo Marina Boillos

² Aunque el TFM son 300 horas, en realidad se han dedicado, aproximadamente, unas 450.