

ESCOLA VIROLAI

**TRABAJO FIN DE GRADO EN ADMINISTRACIÓN DE
SISTEMAS INFORMÁTICOS EN RED**



DESARROLLO DE GESTOR DE TICKETS (CRUD) PARA LA
CREACIÓN, SEGUIMIENTO Y RESOLUCIÓN DE
INCIDENCIAS

Autor: **Mauricio Rey Gallego**

Tutor: Alberto Pozo

Curso académico 2023/2024

Convocatoria de junio

ÍNDICE

ABSTRACT	3
INTRODUCCIÓN.....	4
HIPÓTESIS	5
OBJETIVOS	6
GENERALES	6
ESPECÍFICOS	6
METODOLOGÍA.....	7
HERRAMIENTAS	9
ETAPAS DEL PROYECTO	11
LÓGICA INTERNA DEL PROYECTO.....	12
DESARROLLO DEL PROYECTO	14
CONFIGURACIÓN DEL ENTORNO DE TRABAJO	14
VISUAL STUDIO CODE.....	14
XAMPP	15
CREACIÓN DE LA BASE DE DATOS.....	16
CREACIÓN DE LOS ARCHIVOS	17
RESULTADOS	20
CONCLUSIONES.....	26
BIBLIOGRAFÍA.....	27
ANEXOS	28

ABSTRACT

Este trabajo evalúa la necesidad de sistemas de gestión de incidentes, principalmente en empresas, y describe el diseño y creación de un sistema de gestión de tickets para asuntos de IT realizado con CRUD. Se emplean herramientas como Visual Studio Code, Git (con Github) y MySQL para el desarrollo de este proyecto.

Palabras clave: Gestor de Tickets, Incidencias, Proyecto, Programación, Administración de Sistemas, IT.

This work evaluates the necessity of incident management systems, particularly in companies, and documents the design and creation of a ticket management system for IT incidents with CRUD functionality. The development of this project utilizes tools such as Visual Studio Code, Git (with Github) and MySQL.

Keywords: Ticket Manager, Incidents, Project, Programming, System Administration, IT.

INTRODUCCIÓN

Las experiencias personales del autor son la base de la elección del tema de este trabajo, puesto que, a lo largo de su vida, muchos de sus amigos, familiares y conocidos de todo tipo han requerido su consejo en temas relativos a la informática. El afán de dar una respuesta satisfactoria a todas estas incidencias en el menor tiempo posible ha generado en el desarrollador de este proyecto la inquietud de crear un sistema ordenado de petición, seguimiento y resolución de incidencias. Asimismo, este trabajo viene marcado por un profundo deseo de implicarse en el mundo de la informática desde el punto de vista de la administración de sistemas informáticos, así como del desarrollo de software.

Es importante mencionar que las empresas de cierto tamaño suelen necesitar este tipo de herramientas, con el fin de que las incidencias informáticas que se den en el día a día puedan ser resueltas de forma ordenada, eficiente y ágil. Tal y como menciona García Alonso, "estas aplicaciones normalmente giran alrededor del concepto de *ticket*", teniendo en cuenta que un ticket podría definirse como "la solicitud de servicio que se produce entre dos personas".¹ Es decir, un ticket es, esencialmente, la petición de una persona a otra para que una incidencia determinada sea resuelta o respondida.

Por otro lado, en el mundo de la informática hace falta tener un buen portafolio con diversos proyectos. Sin esto, conseguir un empleo en este campo puede ser auténticamente complicado. Según David Head (fundador de Showspace, una plataforma de creación de portafolios con ayuda de Inteligencia Artificial), "(su) portafolio (le) permitió hacer cosas que otros candidatos no podían hacer, como fue el demostrar lo bueno que podía ser para un trabajo en vez de decirlo" (es decir, sin pruebas).² Teniendo esto en cuenta, así como las muchas conversaciones de este tema que el autor ha tenido con diversas personas con trabajos en el mundo de la informática, parecería que es crucial y necesario incrementar el portafolio personal con proyectos del estilo del que este trabajo trata.

Es por todo lo anteriormente mencionado que el autor ha creído deseable y apropiado desarrollar su propio gestor de tickets ya no solamente para aplicar en su vida privada, sino (sobre todo) para poder implementarlo en pequeñas empresas que así lo requieran. A su vez, existe un gran deseo de adquirir experiencia real y demostrable que pueda ser valorada por terceras personas por su utilidad y/o grado de complejidad.

¹ (Marcos García Alonso, s/f, p. 2)

² (Inglés Original citado en Head, s/f, <https://showspace.so/blog/how-to-use-your-portfolio-to-get-hired>, traducción propia)

HIPÓTESIS

Este trabajo pretende abordar el desarrollo de un gestor de tickets/incidencias implementable en una pequeña empresa. Asimismo, a través de la creación de esta herramienta, se espera ganar experiencia real desarrollando un producto útil, fiable y sencillo de manejar, así como una mejor comprensión de las herramientas utilizadas para la realización del proyecto. Por último, se busca evidenciar que el uso de este gestor de tickets puede aportar unos resultados beneficiosos en aquellas empresas que lo implementen entre sus trabajadores, como pueden ser una mayor eficiencia resolviendo incidencias o un aumento de la productividad, entre otros.

OBJETIVOS

GENERALES

- Desarrollar un gestor de tickets funcional para gestionar incidencias a nivel de empresa.
- Obtener experiencia práctica en el desarrollo de un producto real y útil.
- Adquirir mayor destreza con herramientas como Visual Studio Code, Vim, Git, MySQL, etc.
- Familiarizarme con la metodología “agile”.
- Aprender a ejecutar productos en un servidor propio.
- Acostumbrarse a utilizar control de versiones en el desarrollo de un software.
- Experimentar con la automatización de procesos.

ESPECÍFICOS

- Tener un mayor conocimiento de Git y Github:
 - Hacer commits, branching y merging apropiadamente.
- Adquirir mayor destreza en el uso de Visual Studio Code y Vim:
 - Utilizar extensiones con el objetivo de mejorar la productividad.
 - Mejorar en lo referente a debugging.
- Gestionar incidencias:
 - Diseñar un sistema de gestión de tickets extremadamente fácil de utilizar por futuros clientes.
 - Implementar funcionalidades de creación, seguimiento y resolución de incidencias.
- Documentación del proyecto:
 - Crear documentación concisa, clara y completa de todo el proyecto.
- Evaluar mejoras:
 - Identificar y reparar fallos del producto.
 - Optimizar el producto lo máximo posible.

METODOLOGÍA

Los productos creados en el campo de la informática suelen tener un gran nivel de complejidad debido a que cuentan con una auténtica multitud de partes móviles. Para entender esto mejor, se puede imaginar la producción de una ópera, en la que hay a diversos grupos de gente abordando las diferentes necesidades del proyecto.

En primer lugar, se necesitaría un libreto donde se hable de una historia. En segundo lugar, haría falta la música, para lo cual tendría que haber un compositor (o varios compositores) que crease un contenido musical y lo conectara con el texto del libreto adecuadamente, así como un grupo de músicos que tocaran/cantaran esta música, que a su vez se dividirían entre orquesta y cantantes, y estos luego se fraccionarían en cada uno de los diferentes instrumentos y voces. Después, haría falta un escenario, con lo que serían necesarios unos trabajadores para que el escenario estuviera operativo. Además, se necesitaría un equipo de marketing, de redes sociales, de recaudación de fondos, un tesorero, un secretario, una persona que se encargue de comprar o alquilar las partituras, etc.

En el caso de un producto informático como podría ser el gestor de incidencias que se abarca en este trabajo, al ser éste sencillo no haría falta más que una persona (el autor) para realizarlo. No obstante, sí que es crucial dividir el trabajo en una serie de tareas específicas, con el objetivo de no perderse ni abrumarse en este proyecto y lograr su consecución. Es por este motivo que se requiere seguir una metodología específica.

En el campo de la informática y el desarrollo de software existen varias metodologías, cada cual más o menos apropiada para diversos proyectos. Sin embargo, antes de pasar a explicar la metodología que el autor usará en este proyecto, es conveniente definir “metodología” en el contexto del desarrollo de software, y es que existen muchas definiciones al respecto.

Las metodologías "son enfoques estructurados que se emplean en los proyectos de desarrollo de software."³ También son "un marco de trabajo (framework) utilizado para estructurar, planificar y controlar el proceso de desarrollo de un software."⁴ Otros autores ponen de manifiesto que estas metodologías "buscan mejorar la productividad, la calidad del código y la colaboración."⁵ Si se desea unir y simplificar todas estas definiciones en una sola, se podría decir que una metodología (en el contexto del desarrollo de software) es un procedimiento estructurado para acometer el desarrollo de un proyecto.

Como se ha dicho, hay varias metodologías existentes, y el autor ha optado por utilizar una metodología “agile”, la cual consiste en agilizar el

³ (Fuente: Indeed.com, s/f)

⁴ (GooApps®, 2022)

⁵ (Inglés Original citado en Harvey, 2022, traducción propia)

proceso de desarrollo (de ahí su nombre), dividiendo el proyecto en diversas tareas, las cuales se llevarán a cabo en orden de mayor a menor importancia, dando resultados tangibles rápidamente y mejorando/adaptando el proyecto de manera frecuente. Es decir, que no será un proyecto cuyos resultados se vean al final, sino que los resultados se estarán observando constantemente y a medida que el proyecto tenga lugar. Es importante darse cuenta de que la metodología agile es muy abierta y se puede aplicar de muchas maneras. Es por ello que agile, más que una metodología, podría considerarse como una filosofía o una forma de trabajo.⁶

El autor ha elegido agile por su flexibilidad, así como por su gran importancia en el mundo del desarrollo de software, y es que esta metodología es una de las más utilizadas en la actualidad.⁷ En cuanto a flexibilidad, siguiendo la filosofía agile se pueden sortear los obstáculos con mayor rapidez, ya que no es un desarrollo lineal. Asimismo, al estar el autor realizando este proyecto sin colaboradores, ve necesario dividir el proyecto en partes más pequeñas que pueda realizar una a una, sin abrumarse por la totalidad del proyecto, y viendo resultados constantemente en cada una de las partes.

Hay que destacar que agile no tiene por qué ser mejor ni peor que otras metodologías, pero sí que es la más apropiada para este proyecto, al tener que realizar una serie de entregas en las que se entienda el funcionamiento del gestor de incidencias, y se compruebe la correcta ejecución de cada una de sus partes.

⁶ (Fuente: Indeed.com, s/f)

⁷ (Nikolaieva, s/f)

HERRAMIENTAS

En esta sección, el autor pasa a enumerar las herramientas utilizadas para la ejecución de este proyecto, dividiéndolas en herramientas físicas y lógicas por un lado, y en lenguajes, protocolos y sistemas de gestión, por otro.

Herramientas físicas:

- Ordenador: en este caso, el autor ha optado por utilizar principalmente Windows 11, pero también una distro de GNU/Linux, Debian 12, ya que es muy ligera y personalizable, a la vez que se adapta a la perfección para realizar todos los pasos de este proyecto. A la hora de probar el producto también se utilizará un ordenador Mac con MacOS Sonoma.
- Servidor: se creará un servidor Apache en el equipo local (con XAMPP). De esta manera, el autor busca una mayor sencillez a la hora poner en marcha el proyecto y testarlo. También, se pondrá en marcha un servidor MySQL en el mismo equipo local.
- Router: necesario para tener conexión a internet y poder buscar la información necesaria. Si se quisiera probar a alojar un servidor en un equipo remoto, habría que abrir ciertos puertos para crear conexiones entre equipos.

Herramientas lógicas:

- Visual Studio Code: software para escribir el código necesario, para crear la plataforma de incidencias.
- Vim: mismo que lo anterior, pero para retoques pequeños de ciertos archivos de texto o código (sólo en Linux/MacOS).
- Nano: mismo que lo anterior (sólo en Linux/MacOS).
- GitHub: plataforma utilizada para alojar los distintos archivos del proyecto y poder realizar un control de versiones adecuado.
- Navegadores de internet: se han utilizado varios para comprobar el funcionamiento de la plataforma en los distintos navegadores disponibles (Google Chrome, Mozilla Firefox, Microsoft Edge, Safari...).
- Microsoft Word: para escribir y dar formato al presente trabajo.
- Gedit: para hacer anotaciones rápidas (sólo en Linux).
- Bloc de notas: para hacer anotaciones rápidas (en Windows 11).

Lenguajes, protocolos y sistemas de gestión:

- HTML: lenguaje para escribir el código del sitio web de la plataforma.

- CSS: lenguaje para dar estilo y mejorar el diseño de la plataforma.
- JS: lenguaje de programación para dotar al sitio web de lógica, con el objetivo de que se puedan ejecutar ciertas acciones.
- PHP: Lenguaje necesario para poder conectar la web con la base de datos MySQL.
- XAMPP: software de gestión de servidores, bases de datos, etc.
- Apache: software necesario para crear el servidor web (gestionado con XAMPP).
- MySQL: sistema de gestión de bases de datos para todo lo relacionado con las bases de datos necesarias en este proyecto (gestionado con XAMPP).
- Git: sistema de control de versiones.

A todo lo anterior, habría que añadir todas las fuentes que el autor ha consultado para la creación del sistema de gestión de incidencias y para el estudio relativo a la creación del mismo. Dichas fuentes pueden consultarse en la sección de bibliografía de este trabajo.

ETAPAS DEL PROYECTO

El proyecto se llevará a cabo siguiendo las siguientes tres etapas, las cuales no serán necesariamente lineales en su temporalidad, sino que tendrán numerosas actualizaciones a medida que se lleve a cabo el proyecto, siguiendo una metodología agile.

Primera Etapa – Puesta en marcha de las herramientas:

- Configurar Visual Studio Code.
- Configurar Git y GitHub.
- Configurar XAMPP.
- Crear servidor MySQL y crear su base de datos.
- Crear servidor Apache.
- Comprobar que los servidores locales funcionan correctamente.

Segunda Etapa – Creación de la plataforma:

- Crear sitio web (HTML, CSS, JS, PHP).
- Asegurarse de que las conexiones de PHP con la base de datos funcionen.

Tercera Etapa – Testeo y mantenimiento de la plataforma:

- Pruebas de funcionamiento.
- Localización y arreglo de errores.
- Optimización.

LÓGICA INTERNA DEL PROYECTO

Antes de empezar a construir el gestor de incidencias, se hace necesario explicar la lógica detrás del mismo. De esta manera, se sabrá exactamente qué se necesita hacer y en qué orden. Por ello, hay que enumerar los aspectos a trabajar para empezar a dar forma al proyecto:

- Base de datos: se almacenarán los datos de las incidencias, por lo que la creación de la base de datos debe ser uno de los primeros pasos.
- Servidores MySQL y Apache: para almacenar la base de datos y el sitio web del proyecto, respectivamente. También deberán ser implementados al principio del proceso.
- Páginas web: se emplearán tres. Una para crear incidencias. Otra para ver las incidencias ya creadas y poder editar y/o borrarlas. Y la última para permitir la edición de las incidencias creadas.

Dentro del apartado de la página web, es necesario reseñar que habrá cuatro tipos de archivos utilizados: .html, .css, .js y .php. Paso a explicar brevemente la función de cada uno de estos archivos:

- Archivos HTML: su función es la de mostrar información en pantalla. Básicamente, son el esqueleto de cualquier página web, donde se muestran datos estáticos tales como el título de la web, lo que hace cada botón (enviar, borrar, editar, etc.), los campos de texto donde introducir información, etc.
- Archivos CSS: su función es embellecer la página web. Por ejemplo, se pueden utilizar para dar ciertos colores a distintos textos, centrar párrafos, poner colores de fondo en botones, etc.
- Archivos JS: estos archivos están escritos en el lenguaje JavaScript. Éste es un lenguaje de programación que permite la inclusión de lógica dentro de una página web. Es decir, con archivos HTML y CSS no se pueden meter instrucciones lógicas en una web, sino que nada más se mete información estática. En cambio, con archivos JS se pueden meter instrucciones como, por ejemplo, cambiar el fondo de pantalla cada vez que se haga click en un botón, mostrar determinada información en pantalla en función de lo que haga el usuario, etc. En este caso, se usarán estos archivos para gestionar cualquier acción del usuario, como por ejemplo enviar los formularios o actualizar la web con los datos actuales. Estos archivos se comunicarán con los archivos PHP.
- Archivos PHP: la función de estos archivos (escritos en el lenguaje PHP) es la de procesar lo añadido en la página web en las bases de datos que tengamos en el servidor. Obviamente, en el diseño de un proyecto con funcionalidad CRUD (Create, Read, Update, Delete), estos archivos son esenciales. Se podría haber optado por dar forma a este proyecto utilizando Flask y Python, por ejemplo, pero el autor considera

que con PHP se puede mover el producto a un entorno de producción de manera más eficiente y seria.

DESARROLLO DEL PROYECTO

En el apartado de Etapas del Proyecto, se indican las etapas desde el punto de vista lógico, pero el autor ha decidido que, aquí, el desarrollo se describirá de manera lineal, paso a paso desde cero hasta la compleción del proyecto. Es decir, por claridad se mostrarán todos los pasos necesarios para la creación del proyecto sin tener en cuenta el orden real en que se llevaron a cabo. El paso de crear una cuenta en GitHub se omitirá para centrarse en el producto en sí, a pesar de que es muy útil poder tener un control de versiones y una copia de seguridad de los archivos en caso de que algo fuera mal.

CONFIGURACIÓN DEL ENTORNO DE TRABAJO

Como ya se dijo anteriormente, el proyecto se llevará a cabo en Windows 11, y se recurrirá a XAMPP, y Visual Studio Code.

VISUAL STUDIO CODE

Se va a la página web de Visual Studio Code y se instala.

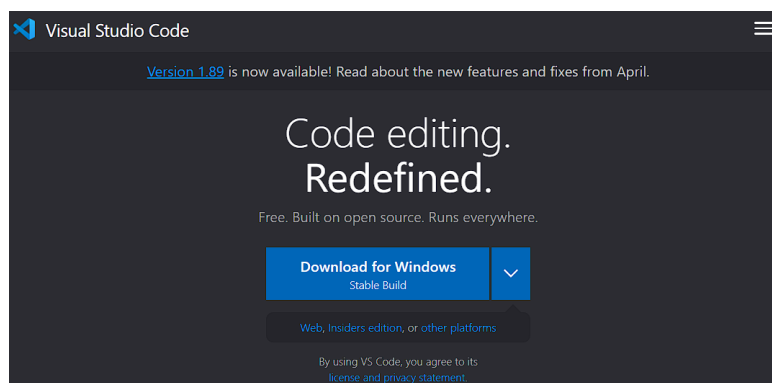


Fig. 1 – Página web de descarga de Visual Studio Code (<https://code.visualstudio.com>) (2024).

Una vez instalado, conviene instalar las extensiones correspondientes de sintaxis para HTML, CSS, JS y PHP.

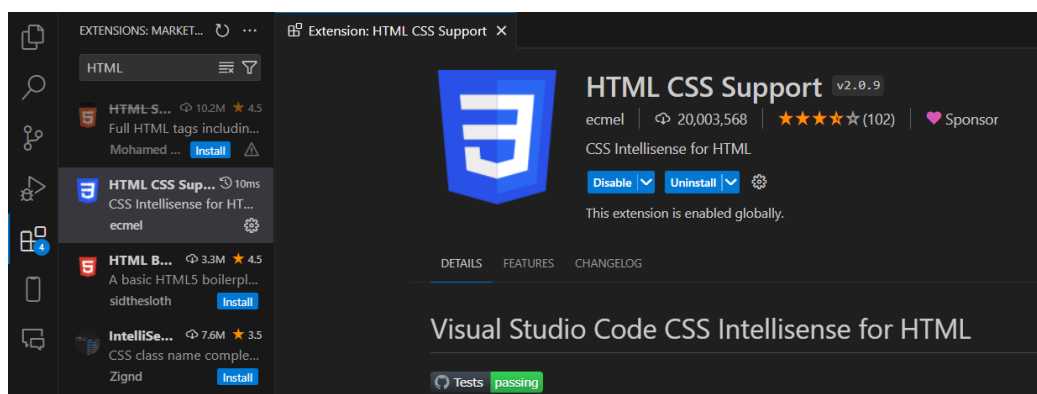


Fig. 2 – Página inicial de Visual Studio Code (2024).

XAMPP

Se hace lo propio con XAMPP. Se accede a su web y se instala.



Fig. 3 – Página web de descarga de XAMPP (<https://www.apachefriends.org/es/index.html>), 2024.

Después se abre XAMPP y hay que asegurarse de que los dos servidores (Apache y MySQL) estén activados y funcionando localmente en el sistema. Si se ha instalado previamente MySQL Workbench o software similar, podría ser que MySQL no se inicie correctamente al principio, sino que haga falta iniciarlo otra vez, tras detener el servicio MySQL previamente. Al final, deberían quedar Apache y MySQL resaltados en verde.

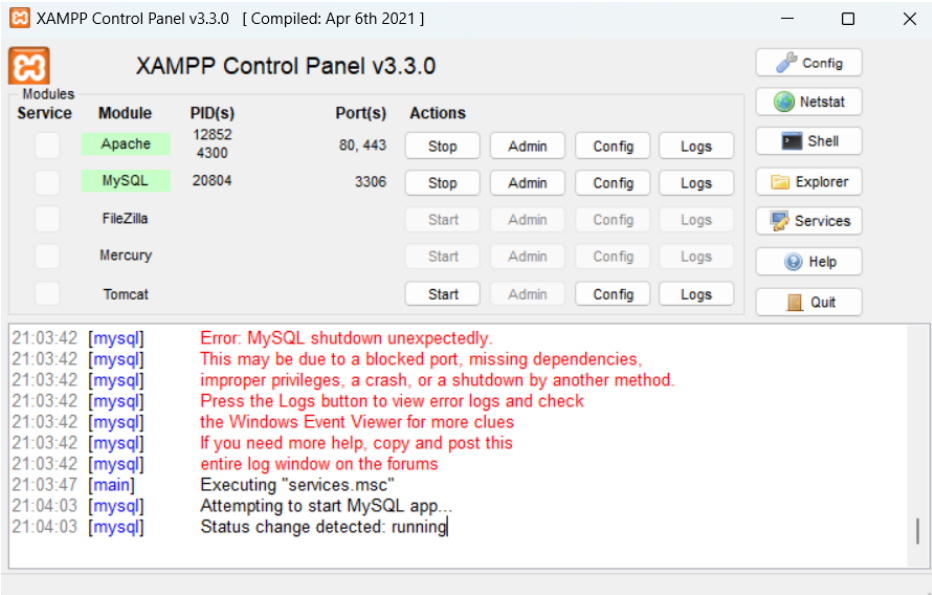


Fig. 4 – Página inicial de XAMPP con los servidores Apache y MySQL iniciados y funcionando (2024).

Con esto ya estaría listo el entorno de trabajo, y se pasaría a la siguiente fase.

CREACIÓN DE LA BASE DE DATOS

Hace falta crear una base de datos para almacenar las incidencias, su número de identificación, su descripción, la hora a la que se crean, etc. Para ello, se accede desde un navegador a la dirección **http://localhost/phpmyadmin** para poder crear la base de datos. Ahí, se crea una nueva base de datos, y se crea una tabla. En el caso de este proyecto, la base de datos que el autor ha creado se llama “db_gestor_incidencias” y la tabla se llama “incidencias”. A continuación, se puede ver el proceso de creación de la tabla.

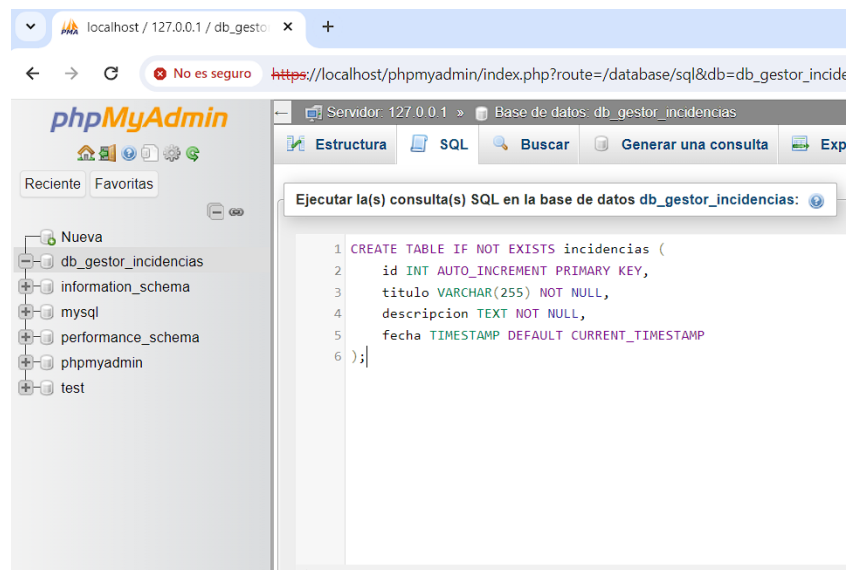


Fig. 5 – página de phpMyAdmin donde se muestra la creación de la tabla dentro de la base de datos (2024).

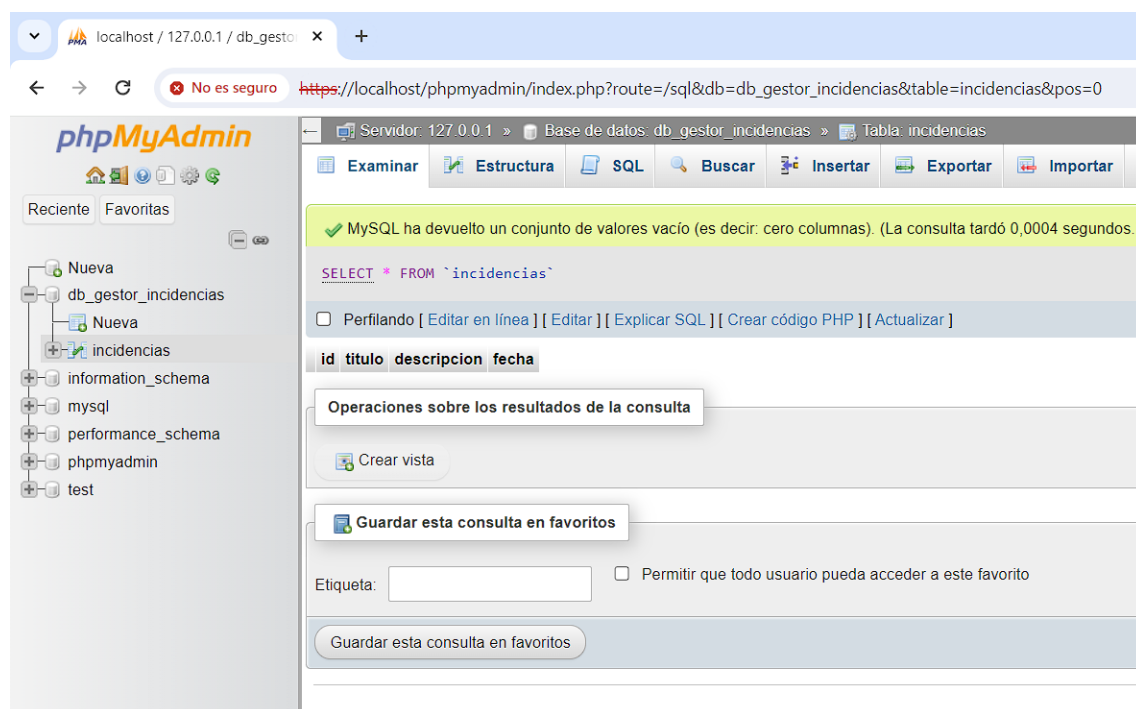


Fig. 6 – página de phpMyAdmin donde se muestra la base de datos y su tabla “incidencias” ya creada (2024).

CREACIÓN DE LOS ARCHIVOS

Una vez están listos tanto el entorno de trabajo como la base de datos y ya están los dos servidores funcionando (Apache y MySQL), se crea una carpeta para contener todos los archivos del proyecto dentro de htdocs, que está a su vez dentro de la carpeta principal de XAMPP en Windows. En este caso, se crea la carpeta “gestor_incidencias”. Esta carpeta tendrá todos los archivos HTML, CSS, JS y PHP necesarios para la creación del gestor de incidencias.

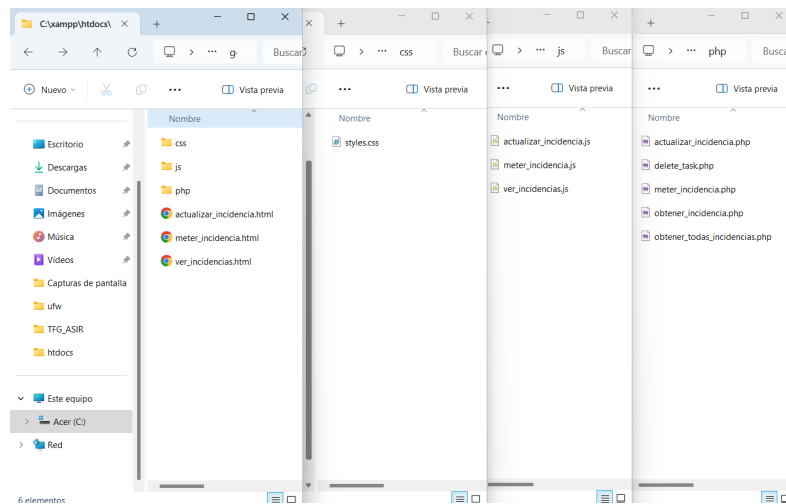


Fig. 7 – creación de la carpeta “gestor_incidencias” dentro de “htdocs”, que está dentro de la carpeta del XAMPP, e inclusión de todos los archivos (en sus respectivas carpetas) que se utilizarán en este proyecto (2024).

Como nota necesaria, hay que indicar que el autor inicialmente optó por utilizar el inglés para dar nombre a los archivos ya que, si el proyecto se fuera a compartir este proyecto con terceras personas en el extranjero, se vería fácilmente para qué sirve cada archivo con sólo leer en nombre. Finalmente, se ha optado por denominar los archivos en español, para ganar claridad en este trabajo, pero no se descarta volver a renombrarlos en inglés en el futuro.

Ahora se puede abrir la carpeta creada (“gestor_incidencias”) dentro de Visual Studio Code y empezar a escribir los archivos necesarios. El autor ha considerado necesario escribir tres páginas HTML. Una para crear incidencias, otra para ver las incidencias creadas y poder elegir editarlas o borrarlas, y otra más para realizar la edición en sí de la tarea. Se puede optar por utilizar una sola página HTML para hacer todo esto, pero el autor piensa que de esta manera se pueden dividir las distintas funciones de manera que se pueda incorporar este gestor de incidencias en un modo en que pueda haber unos usuarios que creen incidencias, y unos administradores quienes, por separado, las puedan ver. También se podrían haber reducido el número de archivos JS y PHP, pero igualmente el autor ha preferido mantener cada función de manera separada en cada archivo.

Teniendo todo esto en cuenta, al final se tendrán doce archivos distintos:

meter_incidencia.html: aquí se pueden añadir incidencias por parte del usuario. Se escriben y se envían al servidor para ser incorporadas a la base de datos.

meter_incidencia.js: este archivo contiene la lógica necesaria para hacer una solicitud POST al servidor (a través de los archivos PHP) con los datos introducidos por el usuario.

meter_incidencia.php: con este archivo se procesan los datos de la incidencia recibida y se inserta en el servidor, dando un mensaje de éxito o de error en formato JSON.

borrar_incidencia.php: este archivo elimina una incidencia especificada previamente por el administrador. Lo hace a través de una solicitud GET presente en ver_incidencias.js. Luego muestra un mensaje de éxito o error en formato JSON.

actualizar_incidencia.html: esta es la página HTML en la que se edita una incidencia. Aquí se ve la incidencia tal y como fue escrita anteriormente en las cajas de texto.

actualizar_incidencia.js: este archivo contiene la lógica necesaria para editar incidencias. Se obtiene la ID de la incidencia con un fetch a obtener_incidencia.php y solicita los datos de ésta en base a esa ID. Después muestra los datos en el formulario y los envía de nuevo, ya actualizados.

actualizar_incidencia.php: gracias a este archivo se reciben los datos actualizados de una incidencia (a través de una solicitud POST previa por parte de actualizar_incidencia.js). Después se actualiza la base de datos y se recibe un mensaje de éxito o error.

obtener_incidencia.php: este archivo es muy importante, ya que, gracias a una solicitud GET de actualizar_incidencia.js, y obtiene los datos de una incidencia en concreto.

obtener_todas_incidencias.php: con este archivo se hace lo mismo que con el anterior, pero ahora es ver_incidencias.js el archivo que hace el fetch a este archivo, con todas las incidencias existentes en la base de datos.

ver_incidencias.html: esta página HTML muestra todas las incidencias presentes en la base de datos, y permite elegir al administrador si desea editar y/o borrar una o varias incidencias.

ver_incidencias.js: este archivo se encarga de la lógica para poder ver todas las incidencias en la misma página, así como para poder editar y/o borrarlas. Se comunica con obtener_todas_incidencias.php y con borrar_incidencia.php.

styles.css: con este archivo se dará el estilo de las páginas .html, y se podrá ofrecer una interfaz más limpia y moderna.

Gracias a todos estos archivos, el gestor de incidencias funcionará con éxito. En la siguiente sección se podrá observar su funcionamiento.

RESULTADOS

Para poder comprobar el funcionamiento del gestor de incidencias, necesitaremos acceder desde un navegador de internet mientras los servidores Apache y MySQL estén en funcionamiento.

En el caso de este proyecto, se accederá a través de http://localhost/gestor_incidencias/meter_incidencia.html para crear incidencias. Asimismo, para ver las ya creadas y tener la posibilidad de editarlas y/o borrarlas, se accederá a través de una dirección distinta: http://localhost/gestor_incidencias/ver_incidencias.html.

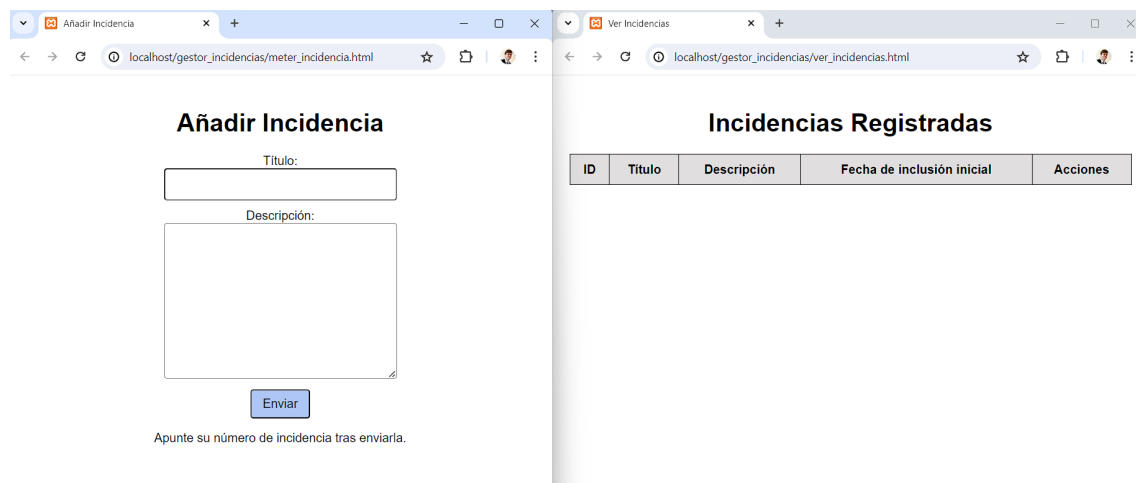


Fig. 8 – páginas para añadir incidencias y ver esas incidencias dentro de mi ordenador local (2024).

Ahora, se puede añadir una incidencia estándar.



Fig. 9 – página de añadir incidencias en proceso de añadir una incidencia (2024).

Cuando se da a “Enviar”, nos sale un mensaje de éxito y un número de incidencia (el número de ID dentro de la base de datos) para que el usuario que ha hecho la incidencia pueda hacer referencia a ella por su número de ID.

The screenshot shows a web browser window with the title 'Añadir Incidencia'. The address bar shows 'localhost/gestor_incidencias/meter_incidencia.html'. A modal message box from 'localhost dice' displays: '¡Incidencia ENVIADA con éxito! Su número de incidencia: 4'. Below this is a text area labeled 'Descripción:' containing the text 'Esta incidencia es la primera incidencia'. At the bottom is a black 'Enviar' button. Below the button, the text 'Apunte su número de incidencia tras enviarla.' is visible.

Fig. 10 – página de añadir incidencias en proceso de añadir una incidencia, con mensaje de éxito (2024).

A continuación, se comprueba si la incidencia es visible desde view_tasks.html y sí, se puede ver con su ID, su título, su descripción y su hora y fecha de inclusión en la base de datos.

The screenshot shows a web browser window with the title 'Ver Incidencias'. The address bar shows 'localhost/gestor_incidencias/ver_incidencias.html'. Below the browser window is a table titled 'Incidencias Registradas'.

ID	Título	Descripción	Fecha de inclusión inicial	Acciones
4	Incidencia de prueba	Esta incidencia es la primera incidencia	2024-05-11 20:04:03	<div>Editar</div> <div>Borrar</div>

Fig. 11 – página de ver incidencias con la incidencia ya creada (2024).

Si se desea editar, se le da a “Editar” y se edita en el nuevo HTML (edit_task.html).

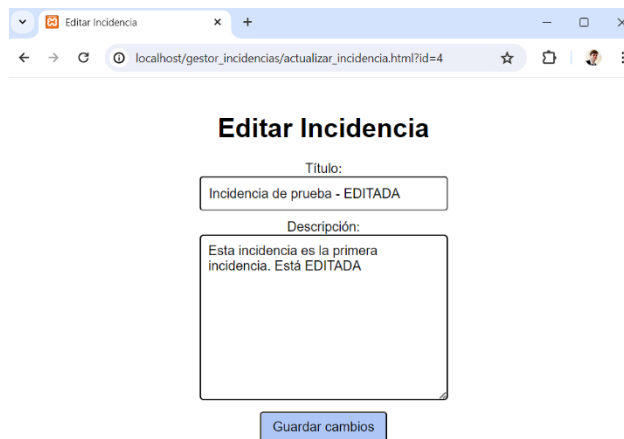


Fig. 12 – página de editar incidencias en proceso de editar una incidencia (2024).

Se da a “Guardar cambios” y se obtiene un mensaje de éxito.

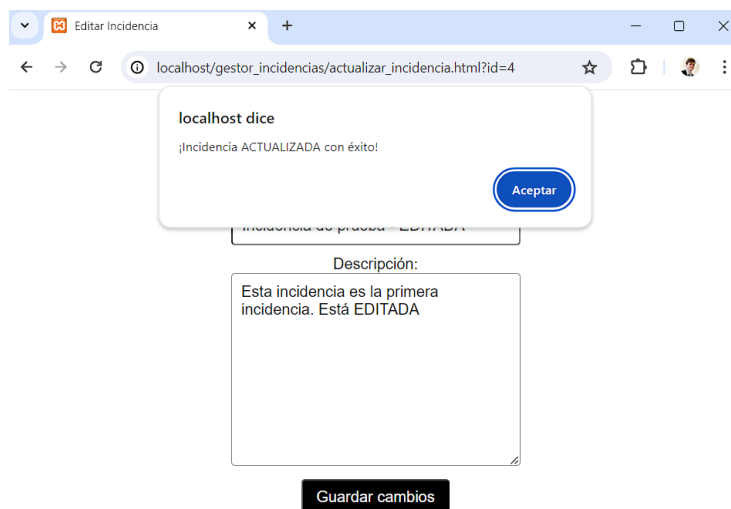


Fig. 13 – página de editar incidencias en proceso de editar una incidencia, con mensaje de éxito (2024).

Se da a “Aceptar” y se verá la incidencia ya actualizada del todo.

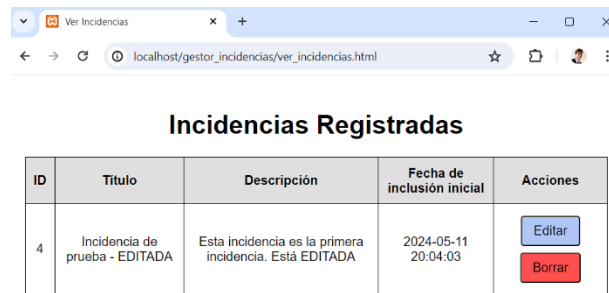


Fig. 14 – página de ver incidencias con la incidencia ya actualizada (2024).

Si ahora se desea borrar la incidencia, no hay más que hacer click en “Borrar” y aceptar el mensaje de advertencia. Después hacer click en aceptar y ver que la incidencia borrada ya no existe.

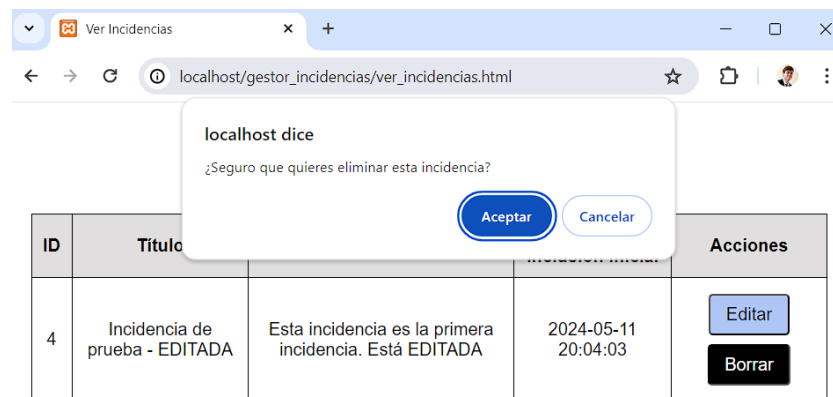


Fig. 15 – página de ver incidencias tras dar al botón “Borrar”, con mensaje de advertencia (2024).

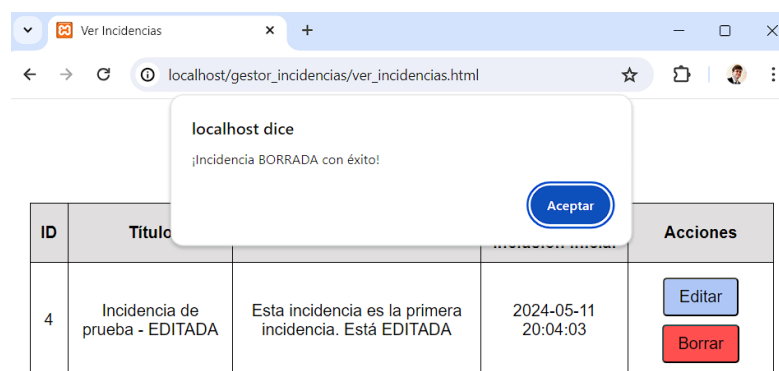
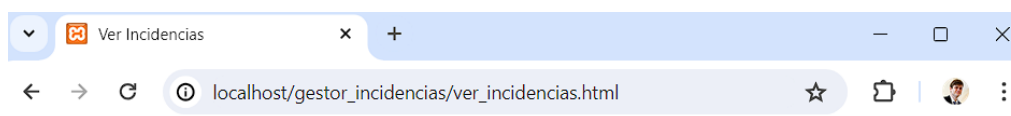


Fig. 16 – página de ver incidencias tras dar al botón “Borrar”, con mensaje de éxito (2024).



Incidencias Registradas

ID	Título	Descripción	Fecha de inclusión inicial	Acciones
----	--------	-------------	----------------------------	----------

Fig. 17 – página de ver incidencias con la incidencia ya borrada (2024).

Se pueden crear muchas incidencias y tener un control óptimo de cada una de ellas, con su ID y su fecha de inclusión.



Incidencias Registradas

ID	Título	Descripción	Fecha de inclusión inicial	Acciones
5	Otra incidencia	Otra incidencia añadida	2024-05-11 20:07:45	<button>Editar</button> <button>Borrar</button>
6	Impresora piso 2	No funciona la impresora del piso 2	2024-05-11 20:08:15	<button>Editar</button> <button>Borrar</button>
7	Clientes no reciben emails	Hay dos clientes que no reciben emails. Los clientes son Julio y María.	2024-05-11 20:08:46	<button>Editar</button> <button>Borrar</button>
8	El ratón no funciona	No me funciona el ratón, estoy escribiendo esto de milagro! Soy Jaime.	2024-05-11 20:09:31	<button>Editar</button> <button>Borrar</button>

Fig. 18 – página de ver incidencias con múltiples incidencias creadas (2024).

Se puede embellecer más la interfaz o se pueden añadir más campos en la base de datos, como, prioridad, progreso, etc. En definitiva, este gestor de incidencias se puede modificar para almacenar prácticamente cualquier información.

Habiéndolo usado durante unos días, el autor considera que un gestor de estas características puede ser muy útil en su vida personal, a la vez que altamente necesario en una empresa que requiera generar incidencias o cualquier otro tipo de información que necesite de una notificación y un

seguimiento. Todo ello, sin recurrir a otros servicios que, aunque puedan ofrecer una mayor funcionalidad, quizá no sean tan sencillos de utilizar. A veces, menos puede ser más, aunque, por supuesto, esto depende del uso que se le quiera dar a un gestor de estas características.

CONCLUSIONES

A lo largo del proceso de creación de este gestor de incidencias se han alcanzado los objetivos planteados al inicio de este trabajo. No solamente se ha conseguido crear el gestor de incidencias de manera artesanal (es decir, sin recurrir a otras opciones ya hechas), sino que se han asimilado una multitud de conocimientos (trabajar con diferentes lenguajes de programación, utilizar GitHub, Visual Studio Code, Vim, Terminal, etc.) así como aprender a seguir una metodología Agile.

Por parte del autor, se considera que la parte más compleja de este proyecto fue la de interconectar todos los archivos y su funcionalidad de manera que todo funcionase correctamente. El concepto teórico era sencillo, pero poner en marcha todos los .js y .php fue, quizá, lo más desafiante.

Asimismo, se entiende que el trabajo está presentado de manera lineal, pero a lo largo del proyecto han tenido lugar muchas pruebas y mucho ir para delante y para detrás de manera que el producto funcionara. Por ejemplo, al principio solamente se hicieron pruebas con una base de datos que sólo aceptaba un campo (el de ID), después se probó la funcionalidad de borrar datos, después de editarlos, etc. Constantemente se intentó que hubiera partes del proyecto que funcionaran, aunque fueran partes muy sencillas, y es que la metodología Agile requiere de ese tipo de prácticas.

En el futuro, se buscará ampliar este producto y convertirlo en algo más grande. También se buscará la forma de hacer pruebas de pentesting para hacer más seguro el producto y conseguir implementarlo de manera profesional en algún lugar que así lo requiera.

Aunque el presente proyecto seguramente tenga cambios significativos en un futuro y estos quizá no reflejen exactamente lo presentado en este trabajo (ya que la versión actual es la primera de posiblemente muchas otras), el repositorio de este proyecto puede encontrarse en:

https://github.com/mreygal/gestor_incidencias

A través de este QR se puede acceder al vídeo de demostración de este gestor de incidencias:



Link hacia: https://youtu.be/3i_NznCheSU

BIBLIOGRAFÍA

- Fazt. (2019, enero 3). *PHP Mysql CRUD* [Video]. YouTube. <https://www.youtube.com/watch?v=pn2v9IPakHQ>
- GooApps®. (2022, octubre 27). Las 5 Mejores Metodologías de Desarrollo de Software. GooApps®. <https://gooapps.es/2022/10/27/las-5-mejores-metodologias-de-desarrollo-de-software/>
- Guía de JavaScript - JavaScript | MDN. (s. f.). MDN Web Docs. <https://developer.mozilla.org/es/docs/Web/JavaScript/Guide>
- Harvey, C. (2022, junio 3). Top 10 programming methodologies. Developer.com. <https://www.developer.com/project-management/10-top-programming-methodologies/>
- Head, D. (s/f). How I've gotten three top tech jobs with only five job applications. Showspace. Recuperado el 29 de febrero de 2024, de <https://showspace.so/blog/how-to-use-your-portfolio-to-get-hired>
- Indeed.com. Recuperado el 31 de marzo de 2024, de <https://es.indeed.com/orientacion-laboral/desarrollo-profesional/metodologias-desarrollo-software-mas-usadas>
- Marcos García Alonso, A. (s/f). *Estudio y desarrollo de un gestor de incidencias portable: UVaHelpDesk*. Uva.es. Recuperado el 29 de febrero de 2024, de <https://uvadoc.uva.es/bitstream/handle/10324/23039/TFG-G2345.pdf;jsessionid=ED61EA160CF8F86DC7B8E7929AD63509?sequence=1>
- Nikolaieva, A. (s/f). 8 best software development methodologies. Uptech.Team. Recuperado el 31 de marzo de 2024, de <https://www.uptech.team/blog/software-development-methodologies>
- PHP Documentation Group. (2024). *PHP Manual*. Recuperado de <https://www.php.net/manual/en/index.php> el 11 de mayo de 2024
- SALM Code. (2022, septiembre 15). *Cómo hacer un CRUD con PHP y MySQL 🐘, paso a paso | Crear, eliminar y editar usuarios con PHP* [Video]. YouTube. <https://www.youtube.com/watch?v=sYaEoNy5OGs>
- Tarea Completo. (2023, February 7).  *CRUD EN PHP y MYSQL DESDE CERO - COMPLETO* [Video]. YouTube. <https://www.youtube.com/watch?v=x4usal92LK8>

ANEXOS

Aquí se incluyen los códigos de los archivos del proyecto. Todos ellos comentados para facilitar su comprensión.

meter_incidencia.html:

```
meter_incidencia.html X
meter_incidencia.html > ...
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <link rel="stylesheet" href="css/styles.css">
6   <meta charset="UTF-8">
7   <meta name="viewport" content="width=device-width, initial-scale=1.0">
8   <title>Añadir Incidencia</title>
9 </head>
10
11 <body>
12   <h1>Añadir Incidencia</h1>
13   <form id="formularioIncidencias">
14     <label for="titulo">Título:</label>
15     <br>
16     <input type="text" id="titulo" name="titulo" required>
17     <br>
18     <label for="descripcion">Descripción:</label>
19     <br>
20     <textarea id="descripcion" name="descripcion" required></textarea>
21     <br>
22     <button class="botonEnviar" type="submit">Enviar</button>
23   </form>
24   <p>Apunte su número de incidencia tras enviarla.</p>
25
26   <!-- Este script es necesario y esencial para que todo funcione: -->
27   <script src="js/meter_incidencia.js"></script>
28 </body>
29
30 </html>
31
```

ver_incidencias.html:

```
ver_incidencias.html X
ver_incidencias.html > ...
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <link rel="stylesheet" href="css/styles.css">
6   <meta charset="UTF-8">
7   <meta name="viewport" content="width=device-width, initial-scale=1.0">
8   <title>Ver Incidencias</title>
9 </head>
10
11 <body>
12   <h1>Incidencias Registradas</h1>
13   <table>
14     <thead>
15       <tr>
16         <th>ID</th>
17         <th>Título</th>
18         <th>Descripción</th>
19         <th>Fecha de inclusión inicial</th>
20         <th>Acciones</th>
21       </tr>
22     </thead>
23     <tbody id="listaIncidencias">
24       <!-- Aquí se listarán las incidencias automáticamente, sacadas de la base de datos, una a una -->
25     </tbody>
26   </table>
27
28   <!-- Este script es necesario y esencial para que todo funcione: -->
29   <script src="js/ver_incidencias.js"></script>
30 </body>
31
32 </html>
33
34
35
```

actualizar_incidencia.html:

```
<> actualizar_incidencia.html X
<> actualizar_incidencia.html > ...
1  <!DOCTYPE html>
2  <html lang="en">
3
4  <head>
5      <link rel="stylesheet" href="css/styles.css">
6      <meta charset="UTF-8">
7      <meta name="viewport" content="width=device-width, initial-scale=1.0">
8      <title>Editar Incidencia</title>
9  </head>
10
11 <body>
12     <h1>Editar Incidencia</h1>
13     <form id="formularioIncidencias">
14         <input type="hidden" id="idIncidencia" name="idIncidencia">
15         <label for="titulo">Título:</label>
16         <br>
17         <input type="text" id="titulo" name="titulo" required>
18         <br>
19         <label for="descripcion">Descripción:</label>
20         <br>
21         <textarea id="descripcion" name="descripcion" required></textarea>
22         <br>
23         <button class="botonEnviar" type="submit">Guardar cambios</button>
24     </form>
25
26     <!-- Este script es necesario y esencial para que todo funcione: -->
27     <script src="js/actualizar_incidencia.js"></script>
28 </body>
29
30 </html>
31
32
```

meter_incidencia.js:

```
JS meter_incidencia.js X
js > JS meter_incidencia.js > ...
1  // Seleccione el formulario del HTML y hago que escuche "envío" y hago una función con ese evento:
2  document.getElementById('formularioIncidencias').addEventListener('submit', function(event) {
3      // Prevengo el funcionamiento por defecto de un submit, que es mandarse:
4      event.preventDefault();
5
6      // Creo variable usando FormData.
7      // Uso FormData porque hay datos diferentes en el formulario (int, text, date, etc.), para evitar errores:
8      var datosFormulario = new FormData(this);
9
10     // Envío datos del formulario a meter_incidencia.php y pido que el .php haga un POST:
11     fetch('php/meter_incidencia.php', {
12         method: 'POST',
13         body: datosFormulario
14     })
15     // Paso la respuesta del fetch de meter_incidencia.php a formato json y consigo un objeto JavaScript con la respuesta:
16     .then(respuesta => respuesta.json())
17
18     // Saco datos de esa respuesta (que ya es un objeto json) y los muestro en pantalla:
19     .then(datos => {
20         alert(datos.aviso);
21         // Reseteo el formulario tras la operación (haya sido ésta exitosa o no):
22         document.getElementById('formularioIncidencias').reset();
23     })
24     // Si hay algún error, se hace un catch para mostrar qué error es:
25     .catch(error => console.error('Error:', error));
26 });
```

ver_incidencias.js:

```
JS ver_incidencias.js X
js > JS ver_incidencias.js > ...

1 // Hago que este .js escuche cuando se carga todo el HTML con DOMContentLoaded, y hago una función:
2 document.addEventListener('DOMContentLoaded', function() {
3 // Envío datos del formulario a obtener_todas_incidencias.php para conseguir las incidencias y ...
4 // ... luego crear código HTML que se "incruste" en la página:
5 fetch('php/obtener_todas_incidencias.php')
6 // Convierto lo resultante a objeto json (por algún motivo no funciona que el .php ya lo pase así) y lo paso a una función:
7 .then(respuesta => respuesta.json())
8 .then(datos => {
9 // Creo un const que cree la lista de incidencias:
10 const listaIncidencias = document.getElementById('listaIncidencias');
11 // Me aseguro de que lo que haya en la lista esté vacío antes de meter incidencias:
12 listaIncidencias.innerHTML = '';
13
14 // Voy creando en HTML cada fila de los datos obtenidos previamente:
15 datos.forEach(incidencia => {
16 const fila = `
17     <tr>
18         <td class="centrar">${incidencia.id}</td>
19         <td class="centrar">${incidencia.titulo}</td>
20         <td class="centrar">${incidencia.descripcion}</td>
21         <td class="centrar">${incidencia.fecha}</td>
22         <td class="centrar">
23             <button class="espacio botonEditar" onclick="actualizarIncidencia(${incidencia.id})">Editar</button>
24             <button class="espacio botonBorrar" onclick="borrarIncidencia(${incidencia.id})">Borrar</button>
25         </td>
26     </tr>
27 `;
28 // Parte esencial para que vaya creando cada nueva fila:
29 listaIncidencias.innerHTML += fila;
30 });
31 });
32 // Si hay errores en este proceso, mostrará esos errores:
33 .catch(error => console.error('Error:', error));
34 });
35
36 // Con esta función redirecciono hacia la página de edición de incidencias tomando el id de la incidencia ...
37 // ... como "ancla", para que se sepa qué se está editando. Quizá no sea una forma de redirección ...
38 // ... excesivamente segura ya que muestra código MySQL en la barra de navegación, pero es funcional:
39 function actualizarIncidencia(taskId) {
40     window.location.href = `actualizar_incidencia.html?id=${taskId}`;
41 }
42
43 // Función para borrar una incidencia que se seleccione, tomando su id:
44 function borrarIncidencia(taskId) {
45     // Hago que salga una ventana de aceptar o cancelar. Si se cancela, se para el if, si se acepta, continúa:
46     if (confirm('¿Seguro que quieres eliminar esta incidencia?')) {
47         // Me comunico con la base de datos a través de borrar_incidencia.php y selecciono una incidencia...
48         // ... en base a su id para borrarla con el method DELETE:
49         fetch('php/borrar_incidencia.php?id=${taskId}', { method: 'DELETE' })
50         // Transformo la respuesta (lo llamo respuesta2 para que funcione y no se mezcle con lo anterior) a json y obtengo el mensaje de esos datos:
51         .then(respuesta2 => respuesta2.json())
52         .then(datos2 => {
53             alert(datos2.avisos);
54             // Actualizo la lista después de borrar la incidencia:
55             location.reload();
56         })
57         // Si hay errores previos, se muestran:
58         .catch(error => console.error('Error:', error));
59     }
60 }
61
62
```

(Continúa en la siguiente página)

actualizar_incidencia.js:

```
JS actualizar_incidencia.js X
js > JS actualizar_incidencia.js > ...
1 // Hago que este .js escuche cuando se carga todo el HTML con DOMContentLoaded, y hago una función:
2 document.addEventListener('DOMContentLoaded', function() {
3     // Creo una const que tenga el valor de los parámetros actuales de la URL actual, la URL de edición con su id específica:
4     const parametrosURL = new URLSearchParams(window.location.search);
5     // Creo otra const con el id de la incidencia, sacándoselo a los parámetros anteriores:
6     const idIncidencia = parametrosURL.get('id');
7
8     // Llamo a obtener_incidencia.php y busco por la id específica:
9     fetch('php/obtener_incidencia.php?id=${idIncidencia}')
10     // Creo objeto json con los datos:
11     .then(respuesta => respuesta.json())
12     // Paso los datos json como argumento a la función y obtengo id, título y descripción ...
13     // ... y nada más, porque no me hace falta la fecha:
14     .then(datos => {
15         document.getElementById('idIncidencia').value = datos.id;
16         document.getElementById('titulo').value = datos.titulo;
17         document.getElementById('descripcion').value = datos.descripcion;
18     })
19
20     // Si hay errores en el proceso, los muestro aquí:
21     .catch(error => console.error('Error:', error));
22 });
23
24 // Seleccione el formulario del HTML y hago que escuche "submit" y hago una función con ese evento:
25 document.getElementById('formularioIncidencias').addEventListener('submit', function(event) {
26     event.preventDefault();
27     // Creo variable usando FormData.
28     // Uso FormData porque hay datos diferentes en el formulario (int, text, date, etc.), para evitar errores:
29     var datosFormulario = new FormData(this);
30
31     // Envío datos del formulario a actualizar_incidencia.php y pido que el .php haga un POST:
32     fetch('php/actualizar_incidencia.php', {
33         method: 'POST',
34         body: datosFormulario
35     })
36     // Paso la respuesta del fetch de actualizar_incidencia.php a formato json y consigo un objeto JavaScript con la respuesta:
37     .then(respuesta2 => respuesta2.json())
38     // Saco datos de esa respuesta (que ya es un objeto json) y los muestro en pantalla:
39     .then(datos2 => {
40         alert(datos2.avisos);
41         // Redirijo a view_tasks.html para poder ver la incidencia actualizada (y el resto de incidencias que haya):
42         window.location.href = 'ver_incidencias.html';
43     })
44     // Si hay errores, los muestro aquí:
45     .catch(error => console.error('Error:', error));
46 });
47
48
```

(Continúa en la siguiente página)

meter_incidencia.php:

```
meter_incidencia.php X
php > meter_incidencia.php
1 <?php
2
3 // Variables para cada aspecto del mysqli:
4 $server = "localhost";
5 $user = "root";
6 // Estoy en modo local, así que no hay contraseña por defecto:
7 $pass = "";
8 $db = "db_gestor_incidencias";
9
10 // Hago la conexión:
11 $conexion = new mysqli($server, $user, $pass, $db);
12
13 // Verifico que la conexión tuvo lugar con connect_error (de mysqli):
14 if ($conexion->connect_error) {
15     die("Conexión fallida: " . $conexion->connect_error);
16 }
17
18 // Creo variables y obtengo los datos de la solicitud POST enviada por meter_incidencia.js
19 $titulo = $_POST['titulo'];
20 $descripcion = $_POST['descripcion'];
21
22 // Creo una variable y almaceno en ella los datos de la consulta SQL para meter la incidencia:
23 $consulta = "INSERT INTO incidencias (titulo, descripcion) VALUES ('$titulo', '$descripcion')";
24
25 // Si la consulta tiene éxito, ejecuto if. SI no, ejecuto else:
26 if ($conexion->query($consulta) === TRUE) {
27     // Cojo el id de la última incidencia metida con el "insert_id" de mysqli:
28     $idUltimaVez = $conexion->insert_id;
29     // Doy mensaje de éxito y almaceno si el éxito es true:
30     $mensaje = array('exito' => true, 'aviso' => "¡Incidencia ENVIADA con éxito!\nSu número de incidencia: $idUltimaVez");
31     // Lo paso a json:
32     echo json_encode($mensaje);
33 } else {
34     // Doy mensaje de error y almaceno si el éxito es falso. El mensaje de error incluye la consulta hecha y su error:
35     $mensaje = array('exito' => false, 'aviso' => 'ERROR enviando incidencia: ' . $consulta . '<br>' . $conexion->error);
36     // Lo paso a json:
37     echo json_encode($mensaje);
38 }
39
40 // Cierro la conexión:
41 $conexion->close();
42
43 ?>
44
```

(Continúa en la siguiente página)

borrar_incidencia.php:

```
borrar_incidencia.php X
php > borrar_incidencia.php
1  <?php
2
3  // Variables para cada aspecto del mysqli:
4  $server = "localhost";
5  $user = "root";
6  // Estoy en modo local, así que no hay contraseña por defecto:
7  $pass = "";
8  $db = "db_gestor_incidencias";
9
10 // Hago la conexión:
11 $conexion = new mysqli($server, $user, $pass, $db);
12
13 // Verifico que la conexión tuvo lugar con connect_error (de mysqli):
14 if ($conexion->connect_error) {
15     die("Conexión fallida: " . $conexion->connect_error);
16 }
17
18 // Creo variable que consiga la id de la incidencia específica solicitada por ver_incidencias.js:
19 $idIncidencia = $_GET['id'];
20
21 // Creo la consulta para borrar la incidencia que tenga un id que concuerde con la anterior:
22 $consulta = "DELETE FROM incidencias WHERE id = $idIncidencia";
23
24 // Si la consulta tiene éxito, ejecuto if. Si no, ejecuto else:
25 if ($conexion->query($consulta) === TRUE) {
26     // Doy mensaje de éxito y almaceno si el éxito es true (si sale el if):
27     $mensaje = array('exito' => true, 'aviso' => '¡Incidencia BORRADA con éxito!');
28     echo json_encode($mensaje);
29 } else {
30     // Doy mensaje de error y almaceno si el éxito es false (si sale el else, en definitiva):
31     $mensaje = array('exito' => false, 'aviso' => 'ERROR borrando incidencia: ' . $conexion->error);
32     echo json_encode($mensaje);
33 }
34
35 // Se cierra la conexión:
36 $conexion->close();
37
38 ?>
39
```

(Continúa en la siguiente página)

actualizar_incidencia.php:

```
actualizar_incidencia.php X
php > actualizar_incidencia.php
1  <?php
2
3  // Variables para cada aspecto del mysqli:
4  $server = "localhost";
5  $user = "root";
6  // Estoy en modo local, así que no hay contraseña por defecto:
7  $pass = "";
8  $db = "db_gestor_incidencias";
9
10 // Hago la conexión:
11 $conexion = new mysqli($server, $user, $pass, $db);
12
13 // Verifico que la conexión tuvo lugar con connect_error (de mysqli):
14 if ($conexion->connect_error) {
15     die("Conexión fallida: " . $conexion->connect_error);
16 }
17
18 // Creo variables para hacer POST con los datos dados por actualizar_incidencia.js
19 $idIncidencia = $_POST['idIncidencia'];
20 $titulo = $_POST['titulo'];
21 $descripcion = $_POST['descripcion'];
22
23 // Consulta SQL para actualizar una entrada de la base de datos con la información anterior:
24 $consulta = "UPDATE incidencias SET titulo='$titulo', descripcion='$descripcion' WHERE id=$idIncidencia";
25
26 // Si la consulta tiene éxito, ejecuto if. Si no, ejecuto else:
27 if ($conexion->query($consulta) === TRUE) {
28     // Doy mensaje de éxito y almaceno si el éxito es verdadero (si salió el if):
29     $mensaje = array('exito' => true, 'aviso' => '¡Incidencia ACTUALIZADA con éxito!');
30     echo json_encode($mensaje);
31 } else {
32     // Doy mensaje de error y almaceno si el éxito es false (si salió el else):
33     $mensaje = array('exito' => false, 'aviso' => 'ERROR actualizando incidencia: ' . $conexion->error);
34     echo json_encode($mensaje);
35 }
36
37 // Cierro la conexión con la base de datos:
38 $conexion->close();
39
40 ?>
41
42
43
```

(Continúa en la siguiente página)

obtener_incidencia.php:

```
obtener_incidencia.php X
php > obtener_incidencia.php
1  <?php
2
3  // Variables para cada aspecto del mysqli:
4  $server = "localhost";
5  $user = "root";
6  // Estoy en modo local, así que no hay contraseña por defecto:
7  $pass = "";
8  $db = "db_gestor_incidencias";
9
10 // Hago la conexión:
11 $conexion = new mysqli($server, $user, $pass, $db);
12
13 // Verifico que la conexión tuvo lugar con connect_error (de mysqli):
14 if ($conexion->connect_error) {
15     die("Conexión fallida: " . $conexion->connect_error);
16 }
17
18 // Creo una variable y le doy el valor del id pedido por actualizar_incidencia.js:
19 $idIncidencia = $_GET['id'];
20
21 // Consulta SQL para obtener una incidencia concreta con el id anterior:
22 $consulta = "SELECT * FROM incidencias WHERE id = $idIncidencia";
23
24 // Se le da a la nueva variable "resultado" el valor de la consulta ya hecha:
25 $resultado = $conexion->query($consulta);
26
27 // Si en ese resultado, la propiedad num_rows (de mysqli) dictamina que hay al menos una fila ...
28 // ... entonces el if se ejecuta, dando a una nueva variable ese resultado y pasándolo a json:
29 if ($resultado->num_rows > 0) {
30     $incidenciaEspecifica = $resultado->fetch_assoc();
31     echo json_encode($incidenciaEspecifica);
32 // Si no es así, entonces se devuelve un objeto json vacío:
33 } else {
34     echo json_encode(array());
35 }
36
37 // Cierro la conexión:
38 $conexion->close();
39
40 ?>
41
```

(Continúa en la siguiente página)

obtener_todas_incidencias.php:

```
obtener_todas_incidencias.php X
php > obtener_todas_incidencias.php
1  <?php
2
3  // Variables para cada aspecto del mysqli:
4  $server = "localhost";
5  $user = "root";
6  // Estoy en modo local, así que no hay contraseña por defecto:
7  $pass = "";
8  $db = "db_gestor_incidencias";
9
10 // Hago la conexión:
11 $conexion = new mysqli($server, $user, $pass, $db);
12
13 // Verifico que la conexión tuvo lugar con connect_error (de mysqli):
14 if ($conexion->connect_error) {
15     die("Conexión fallida: " . $conexion->connect_error);
16 }
17
18 // Variable con el valor de una consulta SQL que seleccione todo de la tabla "incidencias":
19 $consulta = "SELECT * FROM incidencias";
20 // Se le da a la nueva variable "resultado" el valor de la consulta ya hecha:
21 $resultado = $conexion->query($consulta);
22
23 // Creo un array para almacenar todas las incidencias:
24 $listaDeIncidencias = array();
25
26 // Si en ese resultado, la propiedad num_rows (de mysqli) dictamina que hay al menos una fila ...
27 // ... entonces el if se ejecuta y entramos en el while, donde la nueva variable "nuevaFila" se crea ...
28 // ... dándole cada fila gracias al método fetch_assoc. Entonces se van añadiendo las filas a "listaDeIncidencias":
29 if ($resultado->num_rows > 0) {
30     while($nuevaFila = $resultado->fetch_assoc()) {
31         $listaDeIncidencias[] = $nuevaFila;
32     }
33 }
34
35 // Se pasa la lista a json:
36 echo json_encode($listaDeIncidencias);
37
38 // Se cierra la conexión:
39 $conexion->close();
40
41 ?>
42
43
```

(Continúa en la siguiente página)

styles_css:

```
# styles.css X
css > # styles.css > ...

1  body {
2      font-family: Arial, Helvetica, sans-serif;
3      margin: 0;
4      padding: 22px;
5      text-align: center;
6  }
7
8  h1 {
9      text-align: center;
10 }
11
12 form {
13     width: 300px;
14     margin: 0 auto;
15 }
16
17 input[type="text"],
18 textarea {
19     font-family: Arial, Helvetica, sans-serif;
20     border-radius: 4px;
21     padding: 10px;
22     font-size: 16px;
23     margin-bottom: 10px;
24     box-sizing: border-box;
25     width: 100%;
26 }
27
28 textarea {
29     height: 200px;
30 }
31
32 table {
33     width: 100%;
34     border-collapse: collapse;
35 }
36
37 th {
38     border: 1px solid #000000;
39     text-align: center;
40     padding: 10px;
41     background-color: #e0e0e0;
42 }
43
44 td {
45     border: 1px solid #000000;
46     padding: 10px;
47     text-align: left;
48 }
49
50
51
52 .botonEditar {
53     font-family: Arial, Helvetica, sans-serif;
54     font-size: medium;
55     cursor: pointer;
56     padding: 8px 14px;
57     color: black;
58     background-color: #AEC6F6;
59     border: 1px solid black;
60     border-radius: 4px;
61 }
62
63 .botonBorrar {
64     font-family: Arial, Helvetica, sans-serif;
65     font-size: medium;
66     cursor: pointer;
67     padding: 8px 14px;
68     color: black;
69     background-color: #ff4f4f;
70     border: 1px solid black;
71     border-radius: 4px;
72 }
73
74 .botonEnviar {
75     font-family: Arial, Helvetica, sans-serif;
76     font-size: medium;
77     cursor: pointer;
78     padding: 8px 14px;
79     color: black;
80     background-color: #AEC6F6;
81     border: 1px solid black;
82     border-radius: 4px;
83 }
84
85 button:hover {
86     color: white;
87     background-color: black;
88     transition: 0.3s;
89 }
90
91 .centrar {
92     text-align: center;
93 }
94
95 .espacio {
96     margin: 4px;
97 }
98
```