



Cantera Cerro

Foncanal

Autora: Natividad Fernández Alcalá

Año : 2021

Instituto: IES Al-Ándalus

Ciclo : Desarrollo de aplicaciones multiplataforma

Índice

1.- Introducción.....	3
2.- Análisis del entorno.....	3
3.- Análisis del sistema actual.....	10
4.- Solución propuesta.....	10
5.- Planificación temporal del desarrollo del proyecto.....	11
6.- Estudio de la viabilidad del proyecto.....	12
7.- Documentación del diseño e implementación de la solución adoptada.....	13
8.- Código fuente documentado.....	22
9.- Manual de configuración y funcionamiento de la aplicación.....	61
10. Manual de usuario.....	72
11.- Bibliografía y fuentes de información.....	85

1.- Introducción

Nos encontramos con una empresa que tiene una larga historia profesional pero con una nula presencia en Internet. En la actualidad la empresa está inmersa en un proceso de expansión que afronta con unos mecanismos de control de producción obsoletos e insuficientes para la nueva demanda.

Se necesita el desarrollo de una aplicación web y para que sea más versátil y funcional tendrá otra versión en app.

La aplicación a desarrollar busca, por un lado la actualización y modernización de la empresa a nivel digital y virtual, ademas de poder darla a conocer de una manera atractiva y más accesible por estos medios. Por otro lado facilita la recogida y centralización de datos internos a nivel operativo de control de producción, consumos necesarios para la misma (energía, repuesto, personal...etc.) y gestión de stocks. Con esto se generará una información necesaria y actualizada con cada inclusión de nuevos datos, accesible en cualquier momento y lugar. Requisitos indispensables para la inmediatez en la toma de decisiones a nivel de adaptaciones productivas a los jefes de la empresa.

Igualmente la incorporación de más plantilla obliga a instalar mecanismos que registren las horas de entrada y salida del personal, así como las nuevas incorporaciones y bajas laborales de los trabajador@s y puestos de trabajo ocupados. Situación que hasta ahora se hacía de forma personal por un jefe.

Su funcionamiento ha de ser intuitivo, simple y claro, ya que también habrán de usarla personas con poco manejo en estos sistemas.

Esta aplicación sólo podrá ser utilizada por personal de la empresa dado de alta en la misma.

2.- Análisis del entorno.

Esta aplicación se desarrolla en base a las necesidades de una industria en concreto. Pero tiene un gran potencial para otro tipo de industrias donde se necesitan controlar datos de producción diarios y de gestión de stocks. Y en los que los criterios de control son similares. Con muy poca variación puede adaptarse para uso de otras actividades.

Como vamos a desarrollar una aplicación móvil y una web, resulta indispensable conocer de antemano cuales son los requisitos legales para su funcionamiento, a continuación exponemos información recogida de Internet, en la que se describen las cuestiones a tener en cuenta.

REQUISITOS PARA APPS 1

Además de los requisitos técnicos, a la hora de crear una aplicación también hay que tener en cuenta que hay una serie de normativas que se tienen que cumplir. Desde [Ad&Law](#), firma especializada en asesoramiento fiscal, legal y corporativo de start-ups, advierten que "antes de lanzar una aplicación al mercado se tiene que prestar atención a los requisitos legales que, de no tenerse en cuenta, pueden implicar sanciones". Unos requisitos que se pueden resumir en los siguientes ocho puntos:

1 Permisos, licencia y condiciones de uso

Hay que ser claros y explícitos a la hora de solicitar permisos al usuario para acceder a contactos de su dispositivo, realizar pagos o ceder datos. Además, **es obligatorio desarrollar licencias y condiciones de uso.** En todos los casos no basta con informar al usuario sino que este tiene que aceptar, puesto que en caso de reclamación tendremos una mejor defensa.

2 Derechos propios y de terceros

Es obligatorio disponer de licencias de los recursos que se utilizarán. Para lo cual, hay que leer detenidamente las condiciones, puesto que hay casos en los cuales los recursos excluyen el uso comercial, no pudiéndose ejecutar en aplicaciones. Además, **conviene proteger el contenido para evitar plagios y copias.**

3 Menores

En caso de aplicaciones dirigidas a menores de 14 años, **se tienen que consultar las leyes correspondientes y las obligaciones impuestas.** El motivo es que existe una regulación especial en materia de consumidores y usuarios, protección de datos, derechos de imagen, etc.

4 Funcionalidades lícitas

Igual que en el marketing tradicional, **el que es ilícito al offline, al app también lo es.** Por ejemplo, estimular un ámbito de vida poco saludable, como el consumo excesivo de alcohol u otras sustancias.

5 Privacidad y geolocalización.

La recogida de información del usuario tiene que ser la indispensable para el funcionamiento de la app, y este tiene que tener la posibilidad de configurar la privacidad. Además, **si nuestra aplicación dispone de geolocalización, se tiene que contar con la aceptación del usuario** para poder acceder a ella.

6 Información y cookies.

Es fundamental **informar el usuario de los aspectos regulados en la ley** y mostrar los datos sobre los creadores y sobre quienes hay detrás de la aplicación. También es necesario que el usuario acepte las *cookies* mediante un aviso informativo con la información básica y precisa sobre las mismas, así como los aspectos exigidos por la ley.

7 Markets

Tienen condiciones muy estrictas porque se puedan publicar las aplicaciones. Por lo tanto, hay que cumplir siempre el que piden. De hecho, incluso cumpliendo las condiciones al colgar el app, estas pueden cambiar y hacer que la aplicación no esté disponible para usuarios nuevos. Un ejemplo que suelen alegar los markets para rechazar una app es que su interfaz de usuario es compleja o inferior al que catalogan como "muy buena".

8 Publicidad

Si monetizan una aplicación a través de publicidad, esta tiene que identificarse siempre como tal.

En resumen, desde Ad&Law **recomiendan a los emprendedores que no pasen por alto la normativa legal** puesto que "el incumplimiento de estos requisitos puede generar problemas con la

Administración, la justicia y los usuarios. Además, de poder provocar retrasos en el lanzamiento, las actualizaciones o modificaciones de última hora de la aplicación que no estaban previstos y que pueden suponer un coste añadido al proyecto", aseguran.

Información recogida de www.viaempresa.cat

REQUISITOS PARA APP 2

En muchos casos, los emprendedores pueden pensar que el lanzamiento de un negocio a través de una aplicación para dispositivos móviles puede ser sencillo: *con tener la idea, que me la desarrolle un informático, hable con google store y apple store y listo.* Realmente ese sería el esquema genérico, pero no es tan sencillo, necesitas una guía legal para lanzar una aplicación móvil.

Y no es sencillo, ni desde el punto de vista técnico, ni desde el punto de vista legal. Eso claro, si queremos hacerlo bien. Las cosas se pueden hacer mal, y fuera de la ley, con las consecuentes sanciones económicas, y perjuicio de marca en el mercado.

En este trabajo, desarrollo una guía legal para que una empresa o emprendedor pueda lanzar un negocio a través de una app, con todas las garantías legales. Es importante seguir el siguiente orden:

1º.- Protegiendo la idea de negocio que se quiere lanzar a través de la aplicación.

2º.- Firmando un contrato con el desarrollador informático que proteja también tus derechos, sobre todo, los de propiedad intelectual.

3º.- Revisar condiciones de app store y google store, adaptarse y contactar con ellos para poder comercializar la app.

4º.- En caso de trabajar con comerciales externos o distribuidores, firmar un buen contrato o acuerdo con ellos.

5º.- Desarrollar un contrato cliente que te proteja pero que a su vez, no contrarie los derechos de los consumidores y usuarios.

6º.- Respeto y cumplimiento de las leyes aplicables en todo caso, sobre todo, la protección de datos.

1.- Protección de la idea de negocio

Es evidente que hoy día está casi todo inventado, por ello, aquí no trato de decir que hay que protegerlo todo como si fuera una patente en el Registro de Patentes y Marcas. Sin embargo, cuando iniciamos un proyecto empresarial basado en una idea que hasta la fecha no había sido realizada, es importante que la competencia no conozca todos los detalles, y que además lo sepa lo más tarde posible.

Esto implica que lo deben saber la menor cantidad de gente, y una vez desarrollada la idea con algo de forma, se debería firmar un documento de confidencialidad y plan de desarrollo por todos los implicados. Con la inclusión de indemnizaciones económicas que tendrá que satisfacer quien no cumpla el plan diseñado o quien exporte la idea.

Una vez el proyecto tiene suficiente forma, le daremos protección legal. Con el objetivo básico de que nadie copie la idea, y poder demostrar que es vuestra. Hay dos maneras básicas:

- Inscribirlo en el Registro de la Propiedad intelectual
- Acta notarial

Todas las personas que tengan contacto con el proyecto deberían firmar un *acuerdo de confidencialidad y no competencia*. Proteger las ideas de negocio es el primer paso en nuestra guía legal para lanzar una aplicación móvil.

2.- Contrato o acuerdo con el desarrollador informático que va a ejecutar la app.

Este es uno de los momentos más delicados, por eso se debe tener la mayor diligencia. Primero, hasta tanto sepamos si ese informático o empresa, va a ejecutar tu proyecto, dándole la menor información posible. Y después con la redacción de un acuerdo que proteja todos los intereses de tu compañía.

Puede ser que el desarrollador sea un trabajador asalariado de tu empresa. En ese caso, aunque por defecto legal, los derechos de propiedad intelectual serían de la empresa, es conveniente, remarcarlo en el contrato laboral como una cláusula especial, así como el compromiso de confidencialidad y no competencia.

Otro caso, puede ser el de varios socios: uno es el comercial, otro el administrativo y el otro el informático desarrollador. Este la empresa ya constituida o no, convendría firmar un pacto de socios, indicando todas estas cuestiones: a quien corresponden los derechos de propiedad intelectual, los derechos de explotación económica, qué trabajos va a desarrollar cada socio, qué ocurrirá con esos derechos si se disuelve la empresa o el proyecto... En caso de no este pacto de socios, el informático desarrollador tendrá todos los derechos de propiedad intelectual sobre la app. Y si el desarrollo de la app, la encargamos a una empresa externa, es fundamental firmar un contrato de desarrollo de software o app. Algunas de las cuestiones más importantes que debe recoger son: los derechos morales de propiedad intelectual, los derechos de explotación económica, si se va a desarrollar por fases y en qué plazos, si va a existir un tercero que haga de depositario o escrow, sanciones ante incumplimientos, si va a existir contrato de mantenimiento y actualizaciones, etc.

3.- Condiciones y trámites con Apple Store y Google Store.

Por experiencia de clientes que hemos asesorado, sabemos que esta parte es un tanto tediosa, sobre todo, la primera vez, y más con apple que con google.

Este punto, es más bien, de aspectos técnico – informáticos, sin embargo, aprovechamos para hacer algunos comentarios legales.

Cómo estas dos empresas (google y apple) tienen el monopolio de estos sistemas para utilizar y comercializar aplicaciones móviles, debemos pasar por el aro y aceptar todas sus condiciones legales. En cualquier caso, algunas de ellas son abusivas, y por tanto nulas, lo que supone que no tendrán efecto. Las que destaco a este respecto, son las que obligan a demandarles en los EEUU y con la legislación de EEUU. Estas cláusulas no son válidas, si hubiera algún problema se les puede demandar en España.

Ahora, las condiciones de contenido, protección de datos, diseño, etc, hay que cumplirlas porque si no se hace, pueden darte de baja la aplicación, y aunque, después demandemos, desde la baja hasta la sentencia si se gana y te vuelven a activar la app, pasará mucho tiempo sin tenerla activa, y las pérdidas económicas pueden ser grandes.

4.- Contrato con distribuidores o comerciales

En este punto, quiero abordar el posible empleo de comerciales externos para distribuir la app, ya sea a empresas, a particulares o a otros distribuidores, que en ningún caso sean asalariados de la empresa propietaria de los derechos de propiedad intelectual de la app.

Si se emplea a comerciales externos se deben tener varias precauciones:

- Que quede muy claro por escrito, y en la práctica, que no hay relación laboral. Por tanto, que el comercial, cuenta con sus medios económicos y logísticos para desarrollar el trabajo, y cumple con sus obligaciones fiscales y de seguridad social.
- Que queda muy claro en el contrato, de quien son los derechos de propiedad intelectual, y que va a cumplir con el secreto y confidencialidad en su trabajo.
- Qué información puede o no facilitar sobre el negocio, el software y la aplicación.
- Si existe exclusividad o puede comercializar otros servicios y/o productos.
- La prohibición de competencia actual, y la prohibición de comercializar o copiar la aplicación y negocio que está comercializando en el futuro.
- Normas a respetar en materia de protección de datos (pues su incumplimiento puede afectar a tu marca).
- Y las sanciones en caso de incumplir esto.

Estos son algunos ejemplos que tienen que aparecer en todos los contratos con distribuidores, sin embargo, en muchos casos, tras un estudio particular del caso, puede ser necesario incluir otras cuestiones.

5.- Contrato cliente o condiciones generales de contratación

Si es posible que el usuario pueda descargarse las condiciones generales de contratación sería genial. Y si es posible enviárselas por email, todavía mejor. Otra opción es incluir un link a nuestra web, donde pueda descargar y/o leer las condiciones de uso de la app.

En todo caso, es fundamental que haya leído y aceptado las condiciones antes de poder usar la app. En las condiciones generales de contratación o uso, debemos incluir una gran cantidad de cosas. Aquí destaco lo imprescindible:

- La identificación del titular de la app, nombre, o denominación social, dni o cif, vías de contacto, domicilio, email.
- Condiciones económicas
- Actualizaciones y modificaciones
- Condiciones técnicas del terminal
- Usos prohibidos y usos permitidos
- Información básica de protección de datos: responsable del tratamiento, derechos del usuario, período de conservación de los datos, que datos se van a tratar de los usuarios, etc.
- Derechos de propiedad intelectual
- Sistema de quejas o denuncias
- Resolución de incidencias arbitrales o judiciales.
-

Cómo en el contrato con distribuidores será fundamental un estudio pormenorizado del servicio concreto que se presta a través de la app, para saber si es necesario incluir más cláusulas.

6.- Leyes a tener en cuenta

En la guía legal para lanzar una aplicación móvil, debemos respetar una serie de normas y obligaciones legales. Sin duda, la más importante es la protección de datos. La ley Orgánica de Protección de datos personales y garantía de los derechos digitales. Se debe realizar un análisis de riesgos en el tratamiento de los datos personales, respetar los principios de minimización, licitud, exactitud y transparencia en el trato, y tener muy claro y de manera pública: quien es el responsable del tratamiento, y sus datos de contacto, la finalidad del tratamiento, si hay cesión de datos, y si estos salen del territorio de la Unión Europea, las medidas de seguridad y los derechos que tiene el usuario.

Las apps deben tener a disposición del usuario un contenido muy parecido al de los avisos legales y políticas de privacidad y cookies de las páginas webs.

También la Ley de Condiciones Generales de Contratación, por cuanto el usuario podrá aceptar o no usar la web, pero para hacerlo tendrá que aceptar las condiciones, y por tanto, no podrá negociar las mismas.

El Texto Refundido de la Ley de Propiedad Intelectual, por cuanto las ideas de negocio en general y el software de la app, sobre todo, el código fuente, son propiedad intelectual.

La Ley de Servicios de la Sociedad de la Información y el Comercio Electrónico, como empresa o empresario que presta un servicio a través de medios digitales, te es de aplicación esta ley. Cuya obligación fundamental es la identificación del titular.

Aunque la app que lances, no sea para que a través de la misma contraten tus servicios los usuarios, sí se consideran a efectos legales consumidores y usuarios, y por tanto, le es de aplicación La Ley General para la Defensa de los Consumidores y Usuarios, en especial, la regulación específica de la contratación a distancia.

Información recogida de: www.mctabogados.com

REQUISITOS LEGALES PARA PAGINA WEB

1 EL DOMINIO Y LA MARCA

Elegir un dominio no es tarea fácil, y si además tenemos una marca, lo mejor es que coincidan. Si no tenemos ni lo uno ni lo otro, es conveniente registrar ambos a la vez para tener una buena cobertura legal.

No vale elegir sólo un dominio que esté libre y nos guste. Puede haber terceros con una marca anterior ya existente que podrían quitarnos el dominio en el futuro.

2 ¿QUIÉN SOY?

La Ley de Comercio Electrónico (Ley de Servicios de la Sociedad de la Información y de Comercio Electrónico o LSSI) obliga a identificar al titular de una web salvo en aquellas páginas web puramente personales.

Cualquier web profesional, corporativa o comercial debe identificar apropiadamente a su titular.

3 LOS CONTENIDOS

Una web incluye textos, fotos, videos, imágenes, dibujos, etc. ¿Estás seguro de que puedes incluirlos? Dicho de otra forma: los contenidos de terceros pueden estar sujetos a derechos de autor y hay que pedir autorización. Que los contenidos estén ahora en Internet no significa que se puedan utilizar libremente.

4 ¿A QUIÉN ME DIRIJO?

No es lo mismo que la web esté dirigida a empresas, a usuarios finales o, por ejemplo, a menores de edad. En cada caso las obligaciones legales de tu página web pueden cambiar.

5 ¿QUÉ HAGO EN LA WEB?

La actividad a través de la web puede estar regulada como en el caso de sorteos, concursos, juegos o venta de determinados productos y también puede estar sometida a códigos de autorregulación. En este caso, también hay que tenerlo en cuenta para solicitar las oportunas autorizaciones o incluir las leyendas legales que procedan.

6 SI RECOGES DATOS

Un simple formulario de registro supone una recogida de datos que si no cumple con lo que establece la Ley Orgánica de Protección de Datos (LOPD) puede traer consigo una sanción económica de hasta 60.000 euros y en algunos casos, mucho más.

Cumplir con la LOPD es una de las obligaciones más serias que toda web debe cumplir.

7 ¿ENVÍAS UNA NEWSLETTER O INFORMACIÓN COMERCIAL?

¿A quién se la envías? ¿De dónde has sacado los datos? ¿Son ya clientes o son contactos comerciales? La LOPD y la LSSI establecen obligaciones legales en todos los casos

8 COMERCIO ELECTRÓNICO: ¿VENDES EN TU WEB?

Una plataforma de e-commerce requiere cumplir con la LSSI (la cual obliga, por ejemplo, a proporcionar información previa y detallada a la compra) y también con la legislación sobre consumidores.

El incumplimiento conlleva sanciones importantes. Además, una web sin información y transparencia lleva a la desconfianza de tus visitantes.

9 ¿A QUÉ TERRITORIO TE DIRIGES?

En el caso de que tu web se dirija a un país concreto, en su idioma, tal vez te sea aplicable la legislación de dicho país.

10 LOS PROVEEDORES

Conviene que formalices la relación con tus proveedores (desarrolladores, fotógrafos, de hosting, de contenidos) mediante un contrato en el que especifiques claramente las condiciones, qué está incluido y qué no y otras cuestiones que te pueden ser muy útiles en caso de discrepancia.

Y por supuesto, como decíamos en otro post, evita el “corta y pega”. Los especialistas que saben de esto pueden asesorarte para que no corras ningún riesgo y puedas gozar de una larga y legal vida virtual.

Información recogida de www.ttandem.com

Las administraciones estatal y autonómica disponen de recursos económicos para ayudar a la digitalización de las empresas vía subvenciones o ayudas, en nuestro caso se puede contar con:

- **Ayudas y subvenciones para la digitalización de Pymes en España 2020-2025. Ayudas del Gobierno.**
- **Ayudas para digitalizar Pyme de la Junta de Andalucía.**

3.- Análisis del sistema actual.

En la actualidad la empresa carece de cualquier aplicación que conecte los datos que pueden recogerse. Dispone de datos aislados obtenidos a la hora de realizar contabilidad o se encuentran solo en la mente de los jefes, que con su idea global de la marcha del negocio pueden darle una forma aproximada a la marcha de la producción y de los stocks.

No se puede hacer un análisis fidedigno de la situación, además de que esta se encuentra relegada a los jefes exclusivamente, ya que toda la información la manejan ellos. Con la ampliación de la empresa se necesita comunicación entre los distintos departamentos y una centralización de los datos obtenidos.

Igualmente carece de sistema de control de acceso y salida del personal.

4.- Solución propuesta.

- Tecnologías existentes para el desarrollo:
 1. **Laravel:** Es uno de los frameworks de código abierto más fáciles de asimilar para PHP. Es simple, muy potente y tiene una interfaz elegante y divertida de usar.
 2. **Html:** Es un estándar que sirve de referencia del software que conecta con la elaboración de páginas web en sus diferentes versiones, define una estructura básica y un código (denominado código HTML) para la definición de contenido de una página web, como texto, imágenes, videos, juegos, entre otros.
 3. **Css:** es un lenguaje de diseño gráfico para definir y crear la presentación de un documento estructurado escrito en un lenguaje de marcado.
 4. **JavaScript:** Es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico.
 5. **Php:** es un lenguaje de programación de uso general que se adapta especialmente al desarrollo web.
 6. **Android:** es un sistema operativo móvil basado en núcleo Linux y otros software de código abierto.
 7. **Visual Studio Code:** es un editor de código fuente desarrollado por Microsoft para Windows, Linux y macOS. Incluye soporte para la depuración, control integrado de Git, resaltado de sintaxis, finalización inteligente de código, fragmentos y refactorización de código.

8. **Bootstrap:** Es una biblioteca multiplataforma o conjunto de herramientas de código abierto para diseño de sitios y aplicaciones web. Contiene plantillas de diseño con tipografía, formularios, botones, cuadros, menús de navegación y otros elementos de diseño basado en HTML y CSS, así como extensiones de JavaScript adicionales.

9. **Java:** Java es un lenguaje de programación y una plataforma informática comercializada por primera vez en 1995 por Sun Microsystems.

Materiales necesarios para el desarrollo del proyecto:

- Un ordenador y dos pantallas

Recursos Humanos

- Un programador

5.- Planificación temporal del desarrollo del proyecto.

PLANIFICACIÓN TEMPORAL DEL DESARROLLO DEL PROYECTO

Tabla 1

Entrevistas con cliente para obtener especificaciones del proyecto	Investigación para detectar en el mercado de la solución a adoptar	Diseño de la solución a adoptar	Redacción de los presupuestos	Presentación y aprobación del proyecto por el cliente	Adquisición de software y hardware
Entrevistas con encargados o trabajadores responsables para obtener información del proceso					
25/06/2021	28/06 al 2/07/2021	05 al 09 /07/2021	12 al 16 07/2021	19 al 23 07/2021	26/07 al 9/8/2021

		Puede demorarse la redacción por la respuesta de los proveedores	Condicionada al tiempo de contestación del cliente	Se dotará de teléfono con sistema Android a cada emplead@. Se dotará de ordenador a la oficina en planta. Se implantará red wifi. Fase condicionada a intervención de proveedores.
--	--	--	--	---

Tabla 2

Desarrollo de la aplicación web. Desarrollo de la app	Puesta a punto de las aplicaciones	Fase de pruebas	Formación de usuarios finales	Modificaciones o ampliaciones futuras	Plan de mantenimiento de la aplicación
26/07 al 02/10/2021	04/10 al 06/10/2021	07/10 al 14/10/2021	15/10/2021	A petición del cliente	A contar a partir final fase pruebas
	Se hará trabajando en la empresa.	Se hará trabajando en la empresa.	Se formará a los usuarios actuales.	La modificaciones o ampliaciones futuras derivadas de la ampliación de las necesidades de la empresa serán objeto de estudio y de otro presupuesto.	Como norma general se chequeará la aplicación cada tres meses el primer año, o a demanda del cliente. Después anualmente.,o a demanda del cliente

6.- Estudio de la viabilidad del proyecto.

El sector de las canteras en España es un sector en vías de modernización. Las explotaciones, sobre todo las pequeñas o familiares tienen poco recursos y formación para afrontar cambios a niveles informáticos y virtuales. Están ocupados en producir como hace muchos años, obviando las nuevas tecnologías.

Durante la fase de investigación para detectar en el mercado algún formato que se adecuara a las especificaciones del cliente, comprobamos que ninguna cantera o explotación minera de las contactadas tenían app y su páginas web son meramente informativas para posibles clientes.

Contamos con lo novedoso de una versátil app que permite el uso intraempresa de sus trabajadores, no olvidemos que tratamos con una empresa dedicada a la producción de piedra triturada, ellos aportan en el momento que se producen los distintos consumos, aportes o retiradas de stock. Con este funcionamiento la app nos permite saber en tiempo real cual es el estado de la producción con el número de trabajadores implicados en ella, los consumos realizados en cualquier momento (lo que permite a la vez saber si quedan o no existencias de los mismos).

Además sumará, cuando se añadan los clientes una interacción con el exterior.

Estos usos además son compartidos con la web, pudiendo alternar el intercambio de datos en una aplicación u otra. Con lo que tenemos una atractiva web no sólo para dar a conocer a la empresa, sino interactiva que la hace muy útil para el control y desarrollo de la propia actividad.

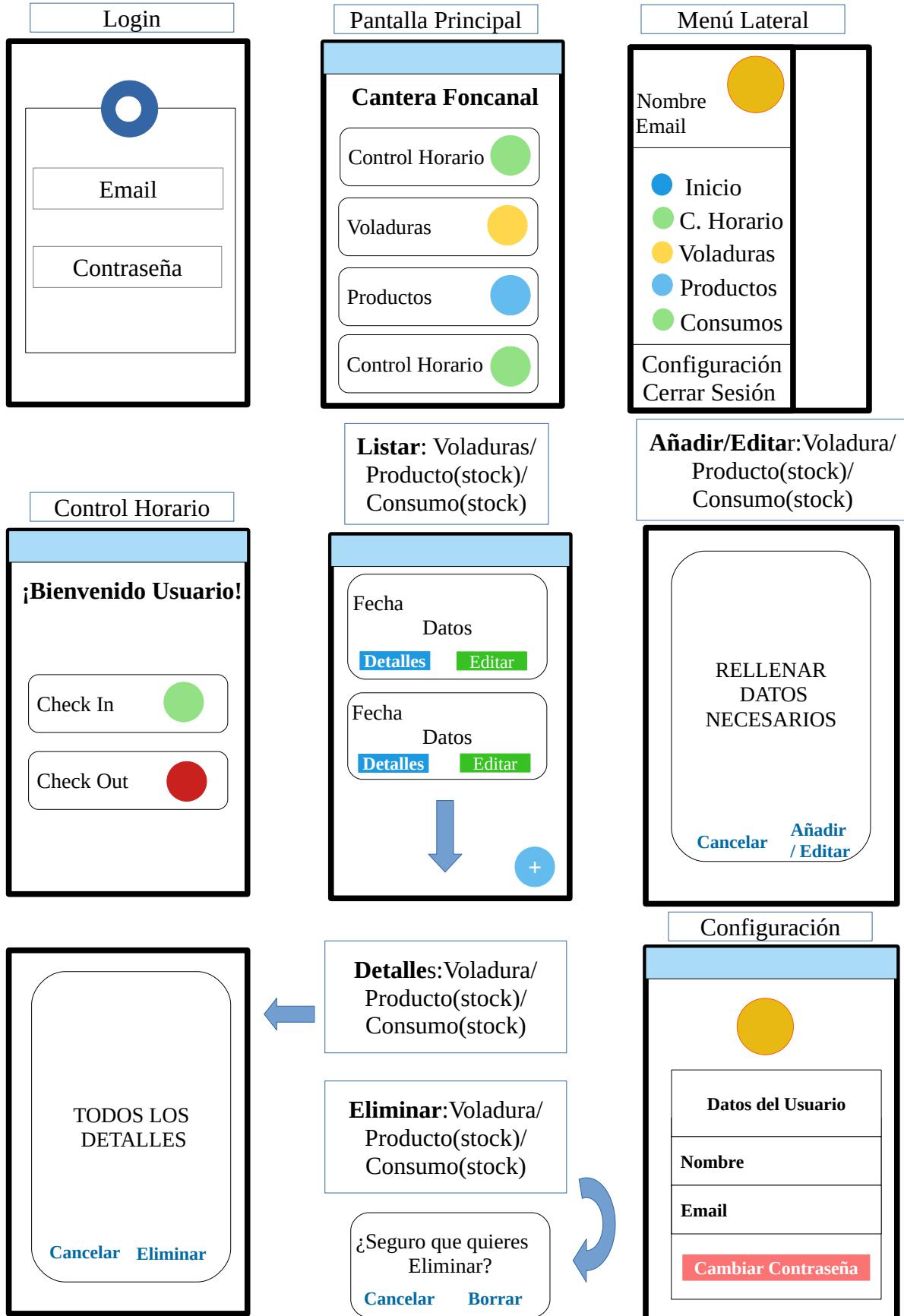
La falta de conectividad de teléfonos e internet; ya que la empresa se encuentra entre montañas, puede ser el mayor de sus inconvenientes. No es una situación habitual, pero podría darse.

También habrá de tenerse en cuenta que puntualmente al trabajad@r puede olvidar el incluir el dato del que es responsable, con el cual el control a realizar de la situación puede no ser real.

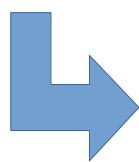
Puede ser que el manejo por parte de personas poco habituadas sea un inconveniente.

7.- Documentación del diseño e implementación de la solución adoptada.

- Diseño de interfaz App Android



Menú
Productos /
Consumos



Menú P/C

- Productos / Consumos (highlighted)
- Stock Pro. / Stock Cons. (highlighted)

- Diseño de interfaz Web

Página de Información

Cantera Foncanal

[Inicio](#) [Producción](#) [Productos](#) [Conócenos](#) [Contacto](#) [¿Eres Cliente?](#)

**Un poco de historia sobre
La Cantera**

!Ubícanos!

Iniciar Sesión

Cantera Foncanal

Iniciar Sesión

Registrarse

Iniciar Sesión

Email

Contraseña

Iniciar Sesión

Registrarse

Cantera Foncanal

Iniciar Sesión

Registrarse

Registrarse

Nombre

Email

Contraseña

Confirmar Contraseña

Registrarse

Pantalla de Inicio



Listar: Empleados / Voladuras /
Productos (stock) / Consumos (stock)

Nombre columnas base de datos		
Datos	●	●
...	●	●
...	●	●
...	●	●

Mostrando registros 1 al 2 **Paginación**

- Editar
- Ficha/ Detalles
- Eliminar

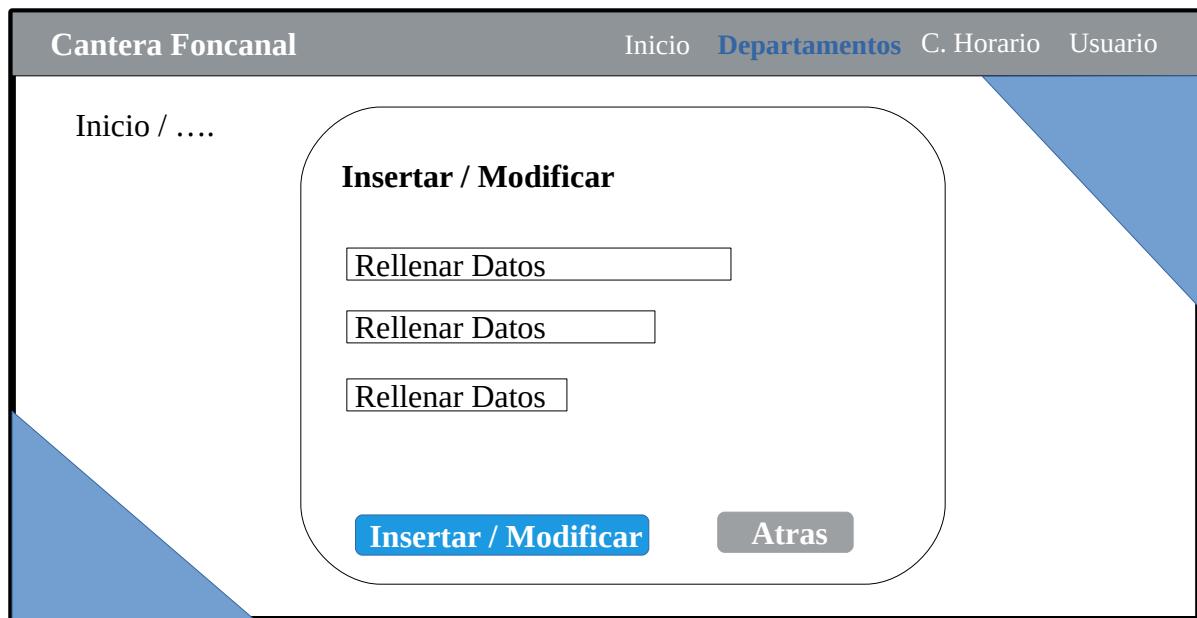
Insertar / Modificar: Empleados / Voladuras /
Productos (stock) / Consumos (stock)

Cantera Foncanal

Inicio Departamentos C. Horario Usuario

Inicio /

Insertar / Modificar



Detalles / Ficha: Empleados / Voladuras /
Productos (stock) / Consumos (stock)

Cantera Foncanal

Inicio Departamentos C. Horario Usuario

Inicio /

Inf.

...

...

...

Ficha Técnica



Eliminar: Empleados / Voladuras /
Productos (stock) / Consumos (stock)

Cantera Foncanal

Inicio Departamentos C. Horario Usuario

Inicio /

4 R

¿Seguro que quieres Eliminar?

Aceptar Cancelar

Nombre columnas base de datos

Datos	● ● ●
...	● ● ●
...	● ● ●
...	● ● ●

Mostrando registros 1 al 2

Paginación

Control Horario

Cantera Foncanal

Inicio Departamentos C. Horario Usuario

Inicio /

> Si Control Horario

4 Registros

Buscar

Nombre columnas base de datos

Datos
Datos

Mostrando registros 1 al 2

Paginación

> No Control Horario

4 Registros

Buscar

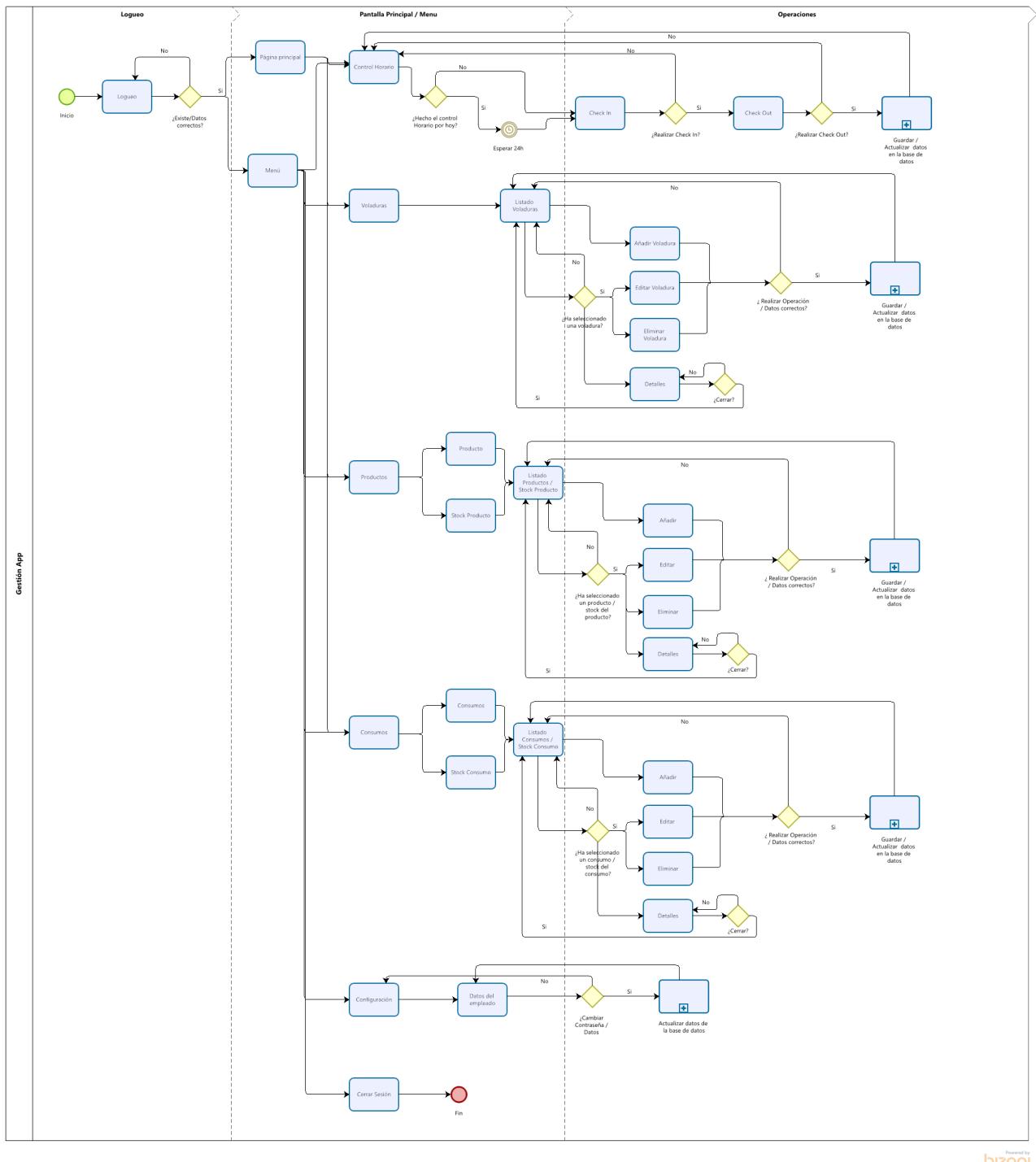
Nombre columnas base de datos

Datos
Datos

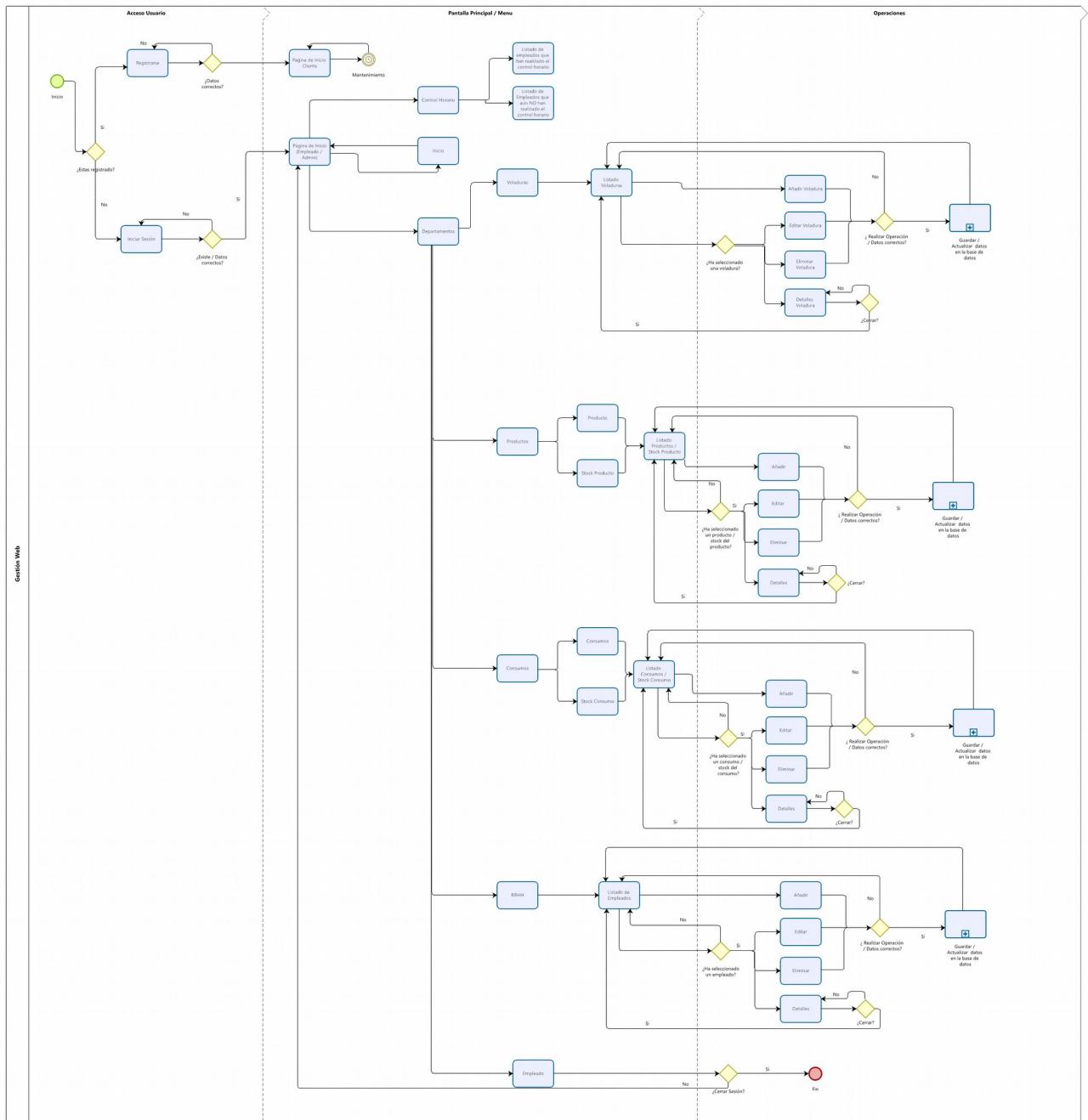
Mostrando registros 1 al 2

Paginación

- Diseño de Flujo de Datos en Android



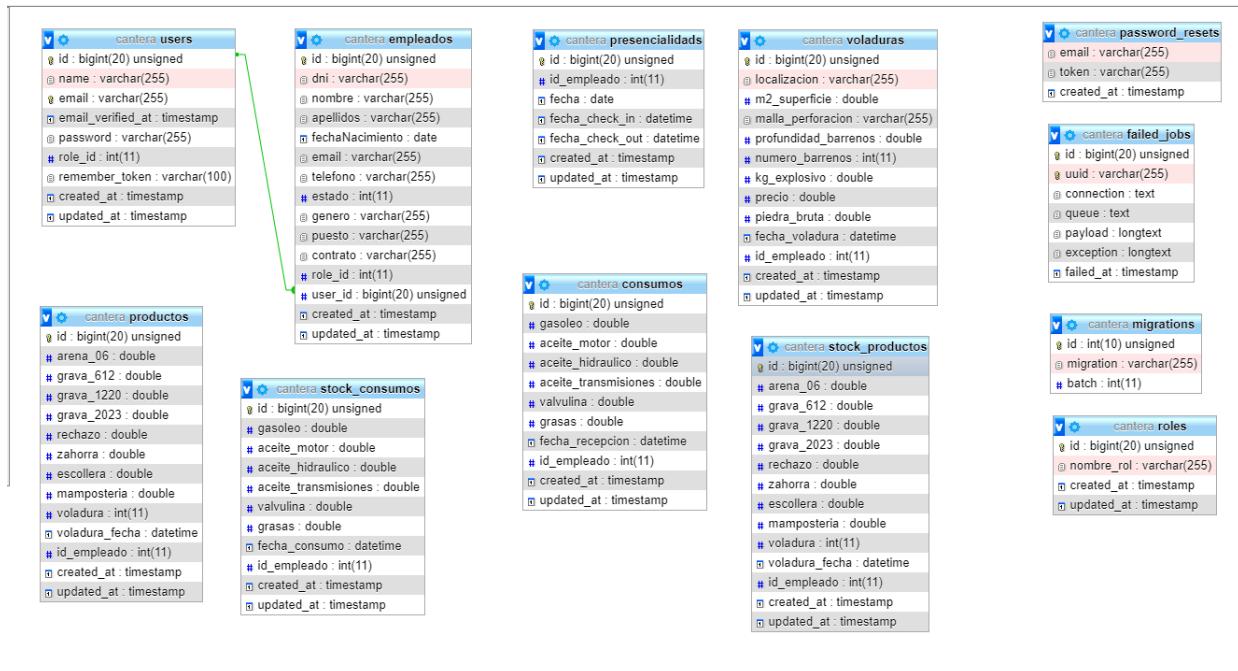
- Diseño de Flujo de Datos en Web



- Diagrama de la Base de Datos Cantera

Las tablas son creadas desde laravel, mediante el uso de migraciones.

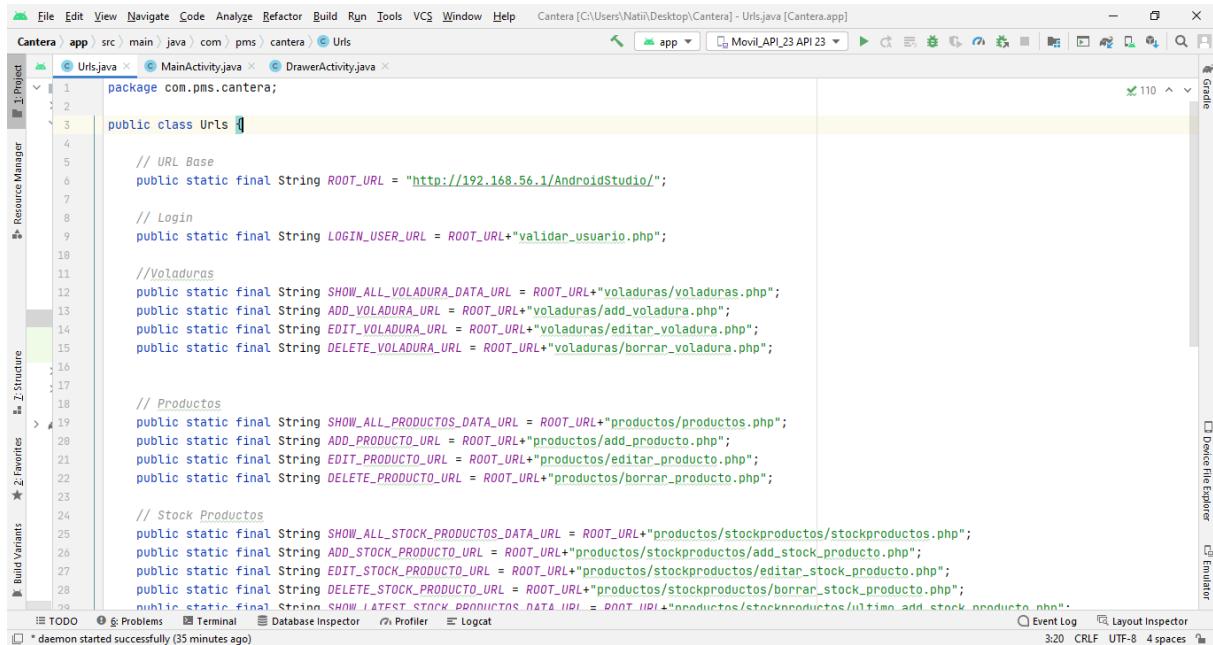
¿Que son las migraciones? - En Laravel, se dice que las migraciones son un control de versiones de nuestra base de datos, pero en realidad son más que eso. Este nos permite crear tablas, establecer relaciones, modificarlas y por supuesto eliminarlas, y todo esto mediante la consola de comandos.



8.- Código fuente documentado

Código fuente Android

1.-Urls: Es la clase que recoge todas las direcciones a los documentos .php, que serán los encargados de actualizar, insertar, eliminar o buscar sobre los datos que le hayamos enviado.



The screenshot shows the Android Studio interface with the URLs.java file open in the main editor. The code defines static final strings for various URLs related to user login, product management, and stock management, all relative to a base URL. The Android Studio interface includes toolbars, a navigation bar, and various panels like Device File Explorer and Layout Inspector.

```
package com.pms.cantera;

// URL Base
public static final String ROOT_URL = "http://192.168.56.1/AndroidStudio/";

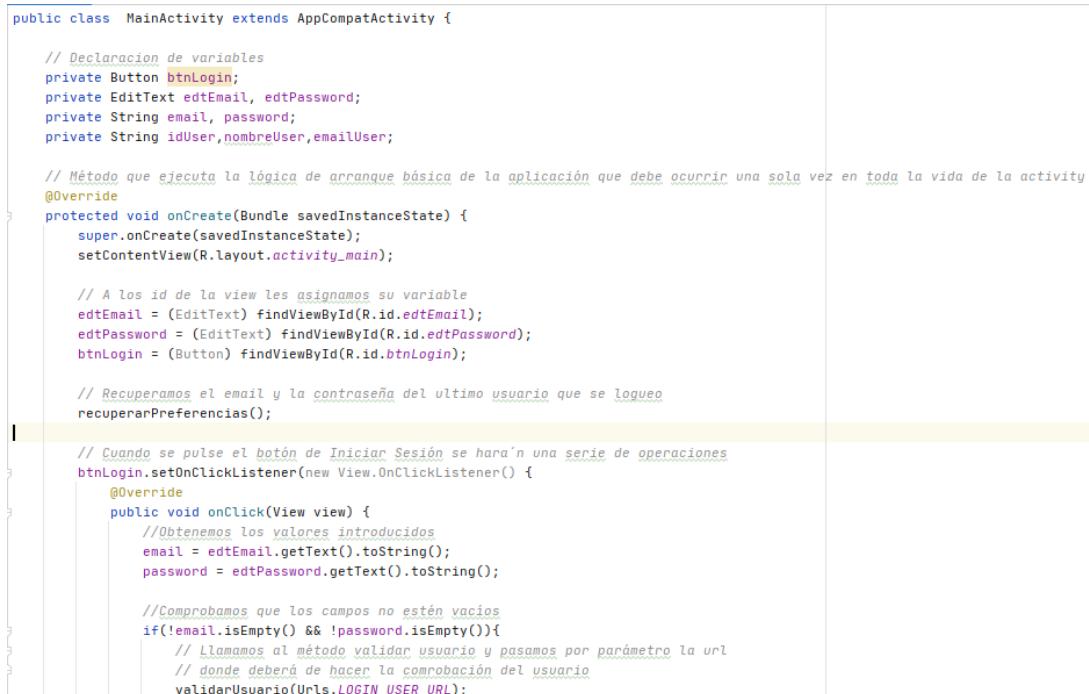
// Login
public static final String LOGIN_USER_URL = ROOT_URL+"validar_usuario.php";

//Voladuras
public static final String SHOW_ALL_VOLADURA_DATA_URL = ROOT_URL+"voladuras/voladuras.php";
public static final String ADD_VOLADURA_URL = ROOT_URL+"voladuras/add_voladura.php";
public static final String EDIT_VOLADURA_URL = ROOT_URL+"voladuras/editar_voladura.php";
public static final String DELETE_VOLADURA_URL = ROOT_URL+"voladuras/borrar_voladura.php";

// Productos
public static final String SHOW_ALL_PRODUCTOS_DATA_URL = ROOT_URL+"productos/productos.php";
public static final String ADD_PRODUCTO_URL = ROOT_URL+"productos/add_producto.php";
public static final String EDIT_PRODUCTO_URL = ROOT_URL+"productos/editar_producto.php";
public static final String DELETE_PRODUCTO_URL = ROOT_URL+"productos/borrar_producto.php";

// Stock Productos
public static final String SHOW_ALL_STOCK_PRODUCTOS_DATA_URL = ROOT_URL+"productos/stockproductos/stockproductos.php";
public static final String ADD_STOCK_PRODUCTO_URL = ROOT_URL+"productos/stockproductos/add_stock_producto.php";
public static final String EDIT_STOCK_PRODUCTO_URL = ROOT_URL+"productos/stockproductos/editar_stock_producto.php";
public static final String DELETE_STOCK_PRODUCTO_URL = ROOT_URL+"productos/stockproductos/borrar_stock_producto.php";
public static final String SHOW_LATEST_STOCK_PRODUCTOS_DATA_URL = ROOT_URL+"productos/stockproductos/ultimo_add_stock_producto.php";
```

2.- MainActivity: Es la clase que realiza el login del usuario, comprueba si este existe y si no, se notificará de ello.



The screenshot shows the Android Studio interface with the MainActivity.java file open. The code implements the onCreate() method to set up the activity's UI and handle button clicks. It uses findViewById() to get references to the login button and text fields, and then sets an OnClickListener on the button. In the onClick() method, it retrieves the email and password from the text fields, checks if they are empty, and then calls a validateUser() method from the URLs class to perform the login check.

```
public class MainActivity extends AppCompatActivity {

    // Declaracion de variables
    private Button btnLogin;
    private EditText edtEmail, edtPassword;
    private String email, password;
    private String idUser,nombreUser,emailUser;

    // Método que ejecuta la lógica de arranque básica de la aplicación que debe ocurrir una sola vez en toda la vida de la activity
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // A los id de la view les asignamos su variable
        edtEmail = (EditText) findViewById(R.id.edtEmail);
        edtPassword = (EditText) findViewById(R.id.edtPassword);
        btnLogin = (Button) findViewById(R.id.btnLogin);

        // Recuperamos el email y la contraseña del ultimo usuario que se logueo
        recuperarPreferencias();

        // Cuando se pulse el botón de Iniciar Sesión se hará una serie de operaciones
        btnLogin.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                //Obtenemos los valores introducidos
                email = edtEmail.getText().toString();
                password = edtPassword.getText().toString();

                //Comprobamos que los campos no estén vacíos
                if(!email.isEmpty() && !password.isEmpty()){
                    // Llamamos al método validar usuario y pasamos por parámetro la url
                    // donde deberá de hacer la comprobación del usuario
                    validarUsuario(Urls.LOGIN_USER_URL);
                }
            }
        });
    }
}
```

```

    } else {
        // Si están vacíos se lo notificamos al usuario
        Toast.makeText( context: MainActivity.this, text: "No se permiten campos vacíos", Toast.LENGTH_SHORT).show();
    }
}

// Método que comprueba si el usuario existe
private void validarUsuario(String URL) {
    StringRequest stringRequest = new StringRequest(Request.Method.POST, URL, new Response.Listener<String>() {
        @Override
        public void onResponse(String response) {
            // Si el resultado de la consulta no está vacío, significa que el usuario existe
            if(!response.isEmpty()){
                try {
                    // Crea un objeto jsonobject para obtener
                    // los valores de la consulta devuelta
                    JSONObject object = new JSONObject(response);

                    // Le asigno a las variables los valores que quiero obtener
                    idUser = object.getString( name: "id");
                    nombreUser = object.getString( name: "name");
                    emailUser = object.getString( name: "email");

                    //Log.i("ID USUARIO",idUser);
                } catch (JSONException e) {
                    e.printStackTrace();
                }
            } else {
                // Mensaje de error
                Toast.makeText( context: MainActivity.this, text: "El email o la contraseña es incorrecta", Toast.LENGTH_SHORT).show();
            }
        }
    }, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {
            // Detectar posibles errores cambiar por mensaje escrito manual
            Toast.makeText( context: MainActivity.this, error.toString(), Toast.LENGTH_SHORT).show();
        }
    });
    @Override
    protected Map<String, String> getParams() throws AuthFailureError {
        // Creamos una instancia el objeto Map
        Map<String, String> parametros = new HashMap<>();
        // Con put ingresamos los datos a enviar
        parametros.put( key: "email", value: email);
        parametros.put( key: "password", value: password);
        return parametros;
    }
};

// Creamos instancia de la clase RequestQueue
RequestQueue requestQueue = Volley.newRequestQueue( context: this);

```

```

// Creamos instancia de la clase RequestQueue
RequestQueue requestQueue = Volley.newRequestQueue( context: this);
// Agregamos la instancia de nuestro objeto StringRequest, ayuda a procesar las peticiones
requestQueue.add(stringRequest);
}

// Método que guarda los datos del usuario
private void guardarPreferencias() {
    SharedPreferences preferences = getSharedPreferences( name: "preferenciasLogin", Context.MODE_PRIVATE);
    SharedPreferences.Editor editor = preferences.edit();
    editor.putString( key: "email", value: email);
    editor.putString( key: "password", value: password);
    editor.putString( key: "idUser", value: idUser);
    editor.putString( key: "nombreUser", value: nombreUser);
    editor.commit();
}

// Método que permite recuperar los datos del usuario
private void recuperarPreferencias() {
    SharedPreferences preferences = getSharedPreferences( name: "preferenciasLogin", Context.MODE_PRIVATE);
    edtEmail.setText(preferences.getString( key: "email", value: "nati@gmail.com"));
    edtPassword.setText(preferences.getString( key: "password", value: "12345678"));
}

// Método que oculta el teclado
public void ocultarTeclado(View view){
    UIUtil.hideKeyboard( activity: this);
}
}

```

Explicación de Métodos de la clase **MainActivity**.

- **ocultarTeclado()** → Como bien dice el nombre, este método ocultará el teclado cuando pulse por cualquier parte de la pantalla visible.
- **guardarPreferencias()** → Método encargado de almacenar los datos del usuario logueado, como su nombre, email, id y contraseña.

Para poder ser almacenados se hace uso de **SharedPreferences**, es una clase que proporciona un conjunto de métodos que permiten almacenar y recuperar muy fácilmente un par clave/valor, con la particularidad de que sólo contiene datos primitivos. Estos datos se mantienen hasta que se cierra completamente la aplicación.

- **recuperarPreferencias()** → Como hemos dicho en el método anterior la clase **SharedPreferences** sirve para recuperar datos, esto es lo que hacemos en este método, para que el usuario no tenga que estar ingresando su email y contraseña constantemente, llamamos al método **recuperarPreferencias** para que ‘auto rellene dichos campos’.
- **validarUsuario(Url)** → Este método comprueba que los datos ingresados por el usuario existen y son correctos, para ello utilizamos **Volley**, esta es una biblioteca HTTP que facilita y agiliza el uso de redes en apps para Android.

Si el usuario existe, lo redirigimos a una nueva activity y si por lo contrario no existe, se notificará de ello. En esta app no podrá registrarse ya que solo tienen acceso los empleados datos de alta. Los registros de estos empleados se hacen desde la web, una vez que sean datos de alta, desde ahí, tendrán acceso a la app.

Dentro de este mismo método utilizo Map, el enviará los datos que le hayamos pasado para que pueda hacer la consulta desde php.

- **onCreate()** → Es el método que ejecuta la lógica de arranque básica de la aplicación que debe ocurrir una sola vez en toda la vida de la activity.

Dentro de este método buscamos los id asociados en la vista del archivo XML de diseño. Los asignamos a las variables que creamos (deben de ser del mismo tipo, TextView, Button....), para poder hacer uso de ellos más fácilmente.

También se declara cuando un botón ha sido pulsado. Aquí si pulsamos el botón Iniciar Sesión (btnLogin), obtenemos mediante **getText** el valor de los campos, comprobamos que no estén vacíos y llamamos al método **validarUsuario()**, pasamos como parámetro la url que necesitamos.

3.- DrawerActivity → Esta clase es la encargada de mostrar el menú lateral y además de llevar a cabo la navegación entre fragmentos.

```

public class DrawerActivity extends AppCompatActivity {
    // Declaración de variables
    private AppBarConfiguration mAppBarConfiguration;
    private ActivityDrawerBinding binding;
    private TextView tvNombre, tvEmail;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // Obtenemos los datos que se nos envian desde la anterior activity
        String idUsuario = getIntent().getStringExtra("idUsuario");
        String nombreUsuario = getIntent().getStringExtra("nombreUsuario");
        String emailUsuario = getIntent().getStringExtra("emailUsuario");

        // Este código es el encargado de la navegación entre fragment y el menú lateral
        binding = ActivityDrawerBinding.inflate(LayoutInflater.from(this));
        setContentView(binding.getRoot());

        setSupportActionBar(binding.appBarDrawer.toolbar);
        binding.appBarDrawer.fab.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Snackbar.make(view, "Replace with your own action", Snackbar.LENGTH_LONG)
                        .setAction("Action", null).show();
            }
        });
        DrawerLayout drawer = binding.drawerLayout;
        NavigationView navigationView = binding.navView;

        // Cuando Cerrar Sesión del menú sea pulsado, llamará al método logout
        navigationView.getMenu().findItem(R.id.logout).setOnMenuItemClickListener(menuItem -> {logout(); return true;});

        mAppBarConfiguration = new AppBarConfiguration.Builder(

```

```

            R.id.principalFragment, R.id.registro_Entradas_Salidas, R.id.voladurasListFragment, R.id.menuProductosFragment, R.id.menuConsumosFragment, R.id.configuracionFragment)
            .setOpenableLayout(drawer)
            .build();
        NavController navController = Navigation.findNavController(this, R.id.nav_host_fragment_content_drawer);
        NavigationUI.setupActionBarWithNavController(this, navController, mAppBarConfiguration);
        NavigationUI.setupWithNavController(navigationView, navController);

        //Quitamos que los íconos aparezcan en gris
        navigationView.setItemIconTintList(null);

        // Obtengo el id de la cabecera
        View header = ((NavigationView) findViewById(R.id.nav_view)).getHeaderView(0);

        // Asigno a las variables el id de los textview
        tvNombre = (TextView) header.findViewById(R.id.tvNombre);
        tvEmail = (TextView) header.findViewById(R.id.tvEmail);

        // Cambiamos el texto de estos text view por los valores obtenidos
        // en la actividad anterior
        tvNombre.setText(nombreUsuario);
        tvEmail.setText(emailUsuario);
    }

    // Creación de las opciones de menu (esquina superior derecha)
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.drawer, menu);
        return true;
    }

    // Navegabilidad entre los fragment del menú lateral izquierdo
    @Override
    public boolean onSupportNavigateUp() {
        NavController navController = Navigation.findNavController(this, R.id.nav_host_fragment_content_drawer);
        return NavigationUI.navigateUp(navController, mAppBarConfiguration)
                || super.onSupportNavigateUp();
    }
}

```

```

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.drawer, menu);
        return true;
    }

    // Navegabilidad entre los fragment del menú lateral izquierdo
    @Override
    public void onBackPressed() {
        // Limpiamos las preferencias que anteriormente creamos
        Sharedpreferences preferences = getSharedpreferences(name="preferenciasLogin", Context.MODE_PRIVATE);
        preferences.edit().clear().commit();

        // Redirigimos a la actividad de login
        Intent intent = new Intent(getApplicationContext(), MainActivity.class);
        startActivity(intent);
        finish();
    }
}

```

Explicación de Métodos de la clase **DrawerActivity**.

- **onCreate()** → Como hemos dicho anteriormente es el método que ejecuta la lógica de arranque básica de la aplicación que debe ocurrir una sola vez en toda la vida de la activity.

Dentro de esta Activity el método `onCreate()` recoge las variables que enviamos desde `MainActivity`, para mostrar el valor obtenido (nombre y correo electrónico) en el las textview de este nuevo XML.

Además se gestiona la navegabilidad entre fragments del menú. Cada uno de los enlaces del menú nos lleva a un nuevo fragment, excepto Cerrar Sesión.

Cuando Cerrar Sesión sea pulsado llamará al método **logout()** (este método se explica más abajo) y nos llevará de nuevo a `MainActivity`.

- **onCreateOptionsMenu(Menu)** → Método encargado de la funcionalidad del menú superior derecho. En la app aparecerán tres puntos en la posición indicada, si pinchamos en él nos aparecerá un mensaje, puesto que se encuentra en mantenimiento. (Futuras actualizaciones)
- **OnSupportNavigateUp()** → Se crea un objeto **Navigation** que hace referencia al fragment ‘padre’ (hago referencia como fragment padre porque es el que hace de base sobre los demás fragment que los llamaré ‘hijos’), este se enlaza con sus fragment hijos, que son por los que iremos navegando durante la ejecución de la aplicación.
- **logout()** → Este método cierra la sesión del usuario conectado. Utilizamos **SharedPreferences** para que borre todas las preferencias que se hayan guardado durante la ejecución de la aplicación. Creamos un objeto intent para que nos redirija a `MainActivity`.

4.- Carpeta Voladuras:

Dentro de esta carpeta nos encontramos tres clases.

1. **Voladuras:** Definimos el objeto de voladura con sus atributos y métodos necesarios. Creamos las variables privadas y sus métodos get y set para después hacer uso de ellos.

```

public class Voladuras {
    private Integer id;
    private String localizacion;
    private Double m2_superficie;
    private String malla_perforacion;
    private Double profundidad_barrenos;
    private Integer numero_barrenos;
    private Double kg_explotivo;
    private Double precio;
    private Double piedra_bruta;
    private String fecha_voladura;
    private Integer id_empleado;

    public Integer getId() { return id; }

    public void setId(Integer id) { this.id = id; }

    public String getLocalizacion() { return localizacion; }

    public void setLocalizacion(String localizacion) { this.localizacion = localizacion; }

    public Double getM2_superficie() { return m2_superficie; }

    public void setM2_superficie(Double m2_superficie) { this.m2_superficie = m2_superficie; }

    public String getMalla_perforacion() { return malla_perforacion; }

    public String getMalla_perforacion() { return malla_perforacion; }

    public void setMalla_perforacion(String malla_perforacion) {
        this.malla_perforacion = malla_perforacion;
    }

    public Double getProfundidad_barrenos() { return profundidad_barrenos; }

    public void setProfundidad_barrenos(Double profundidad_barrenos) {
        this.profundidad_barrenos = profundidad_barrenos;
    }

    public Integer getNumero_barrenos() { return numero_barrenos; }

    public void setNumero_barrenos(Integer numero_barrenos) {
        this.numero_barrenos = numero_barrenos;
    }

    public Double getKg_explotivo() { return kg_explotivo; }

    public void setKg_explotivo(Double kg_explotivo) { this.kg_explotivo = kg_explotivo; }

    public Double getPrecio() { return precio; }

    public void setPrecio(Double precio) { this.precio = precio; }

    public Double getPiedra_bruta() { return piedra_bruta; }
}

```

2. VoladurasListFragment: Esta clase está enlazada con la clase voladurasAdapter, simplemente ‘listamos’ (está puesto entre comillas porque el listar lo llevan a cabo voladurasListFragment y voladurasAdapter) y añadimos voladuras.

```

public class VoladurasListFragment extends Fragment {
    // Declaración de variables
    private RecyclerView recyclerView;
    private VoladurasAdapter voladurasAdapter;
    private List<Voladuras> voladurasList;
    private FloatingActionButton add_voladura;
    private String idUsuario;

    // Se carga la vista
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                           Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.fragment_voladuras_list, container, false);
    }

    // Inicializamos las clases una vez que se sabe que su jerarquía de vista se ha creado por completo
    @Override
    public void onViewCreated(@NonNull @NotNull View view, @Nullable @org.jetbrains.annotations.Nullable Bundle savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);

        SharedPreferences preferencess = getContext().getSharedPreferences("preferenciasLogin", Context.MODE_PRIVATE);

        // Accedemos al idUsuario desde las preferencias
        idUsuario = preferencess.getString("idUsuario", "id");

        // Asignamos las id a las variables
        recyclerView = view.findViewById(R.id.recyclerView);
        // Establecemos el diseño de los contenidos, es decir, la lista de vistas repetidas en la vista del recyclerView
        recyclerView.setHasFixedSize(true);
        recyclerView.setLayoutManager(new LinearLayoutManager(getContext())); //getActivity()

        // creamos un ArrayList
        voladurasList = new ArrayList<>();
    }
}

```

```

// Leemos todas las voladuras
LoadAllVoladuras();

// Botón añadir voladuras
add_voladura = (FloatingActionButton) view.findViewById(R.id.add_btn_voladura);

// Cuando se pulse el botón añadir voladura, llamará al método addVoladura
add_voladura.setOnClickListener(new View.OnClickListener() {
    @RequiresApi(api = Build.VERSION_CODES.LOLLIPOP)
    @Override
    public void onClick(View view) { AddVoladura(view); }
});

// Método que leerá todas la voladuras existentes en la base de datos
private void LoadAllVoladuras() {
    // Consulta a la base de datos, para que nos devuelva los las voladuras
    JsonArrayRequest request = new JsonArrayRequest(Urls.SHOW_ALL_VOLADURA_DATA_URL, new Response.Listener<JSONArray>() {
        @Override
        public void onResponse(JSONArray response) {
            // Recorremos el array de voladuras
            for(int i = 0; i < response.length(); i++){
                try {
                    // Crea un objeto json, accede a los valores y se lo asigna a las variables
                    JSONObject object = response.getJSONObject(i);
                    Integer id = object.getInt("id");
                    String localizacion = object.getString("localizacion").trim();
                    Double m2_superficie = object.getDouble("m2_superficie");
                    String malla_perforacion = object.getString("malla_perforacion").trim();
                    Double profundidad_barrenos = object.getDouble("profundidad_barrenos");
                    Integer numero_barrenos = object.getInt("numero_barrenos");
                    Double kg_explotivo = object.getDouble("kg_explotivo");
                    Double precio = object.getDouble("precio");
                    Double piedra_bruta = object.getDouble("piedra_bruta");
                    String fecha_voladura = object.getString("fecha_voladura").trim();
                    Integer id_empleado = object.getInt("id_empleado");
                } catch (JSONException e) {
                    e.printStackTrace();
                }
            }
        }
    }, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {
            // Mensaje de error
            Toast.makeText(getApplicationContext(), error.toString(), Toast.LENGTH_SHORT).show();
        }
    });
}

// Método que añade una nueva voladura
@RequiresApi(api = Build.VERSION_CODES.LOLLIPOP)
private void AddVoladura(View view){
    // Instancia del archivo XML de diseño
    View editLayout = LayoutInflater.from(getApplicationContext()).inflate(R.layout.voladura_add, null);
    // Asignamos las id a las variables
    TextView Id = editLayout.findViewById(R.id.add_id);
    EditText Localizacion = editLayout.findViewById(R.id.add_localizacion);
    EditText M2_superficie = editLayout.findViewById(R.id.add_m2_superficie);
    EditText Malla_perforacion = editLayout.findViewById(R.id.add_malla_perforacion);
    EditText Profundidad_barrenos = editLayout.findViewById(R.id.add_profundidad_barrenos);
    EditText Numero_barrenos = editLayout.findViewById(R.id.add_numero_barrenos);
    EditText Kg_explotivo = editLayout.findViewById(R.id.add_kg_explotivo);
    EditText Precio = editLayout.findViewById(R.id.add_precio);
    EditText Piedra_bruta = editLayout.findViewById(R.id.add_piedra_bruta);
    EditText Fecha_voladura = editLayout.findViewById(R.id.add_fecha_voladura);
    TextView Id_empleado = editLayout.findViewById(R.id.add_id_empleado);

    // Ocultamos el teclado
    Fecha_voladura.setInputType(InputType.TYPE_NULL);

    // Cuando se pulse el campo Fecha voladura, llamamos al método showDateTimeDialog
    Fecha_voladura.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) { showDateTimeDialog(Fecha_voladura); }
    });
}

```

```

Double profundidad_barrenos = object.getDouble("profundidad_barrenos");
Integer numero_barrenos = object.getInt("numero_barrenos");
Double kg_explotivo = object.getDouble("kg_explotivo");
Double precio = object.getDouble("precio");
Double piedra_bruta = object.getDouble("piedra_bruta");
String fecha_voladura = object.getString("fecha_voladura").trim();
Integer id_empleado = object.getInt("id_empleado");

// Creación objeto Voladura
Voladuras voladuras = new Voladuras();
voladuras.setId(id);
voladuras.setLocalizacion(localizacion);
voladuras.setM2_superficie(m2_superficie);
voladuras.setMalla_perforacion(malla_perforacion);
voladuras.setProfundidad_barrenos(profundidad_barrenos);
voladuras.setNumero_barrenos(numero_barrenos);
voladuras.setKg_explotivo(kg_explotivo);
voladuras.setPrecio(precio);
voladuras.setPiedra_bruta(piedra_bruta);
voladuras.setFecha_voladura(fecha_voladura);
voladuras.setId_empleado(id_empleado);

// Añadimos las voladuras a un List
voladurasList.add(voladuras);

} catch (JSONException e) {
    e.printStackTrace();
}

// Creación de objeto Adapter, necesitamos por parámetro getContext() que se refiere a la clase
// en la que nos encontramos y a la List creada anteriormente
voladurasAdapter = new VoladurasAdapter(getContext(), voladurasList);
recyclerView.setAdapter(voladurasAdapter);
}, new Response.ErrorListener() {
    @Override
    public void onErrorResponse(VolleyError error) {
        // Mensaje de error
        Toast.makeText(getApplicationContext(), error.toString(), Toast.LENGTH_SHORT).show();
    }
});

```

```

    @Override
    public void onErrorResponse(VolleyError error) {
        // Mensaje de error
        Toast.makeText(getApplicationContext(), error.toString(), Toast.LENGTH_SHORT).show();
    }
});

RequestQueue requestQueue = Volley.newRequestQueue(getApplicationContext());
requestQueue.add(request);

// Método que añade una nueva voladura
@RequiresApi(api = Build.VERSION_CODES.LOLLIPOP)
private void AddVoladura(View view){
    // Instancia del archivo XML de diseño
    View editLayout = LayoutInflater.from(getApplicationContext()).inflate(R.layout.voladura_add, null);
    // Asignamos las id a las variables
    TextView Id = editLayout.findViewById(R.id.add_id);
    EditText Localizacion = editLayout.findViewById(R.id.add_localizacion);
    EditText M2_superficie = editLayout.findViewById(R.id.add_m2_superficie);
    EditText Malla_perforacion = editLayout.findViewById(R.id.add_malla_perforacion);
    EditText Profundidad_barrenos = editLayout.findViewById(R.id.add_profundidad_barrenos);
    EditText Numero_barrenos = editLayout.findViewById(R.id.add_numero_barrenos);
    EditText Kg_explotivo = editLayout.findViewById(R.id.add_kg_explotivo);
    EditText Precio = editLayout.findViewById(R.id.add_precio);
    EditText Piedra_bruta = editLayout.findViewById(R.id.add_piedra_bruta);
    EditText Fecha_voladura = editLayout.findViewById(R.id.add_fecha_voladura);
    TextView Id_empleado = editLayout.findViewById(R.id.add_id_empleado);

    // Ocultamos el teclado
    Fecha_voladura.setInputType(InputType.TYPE_NULL);

    // Cuando se pulse el campo Fecha voladura, llamamos al método showDateTimeDialog
    Fecha_voladura.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) { showDateTimeDialog(Fecha_voladura); }
    });
}

```

```

        public void onClick(View view) { showDateTimeDialog(Fecha_voladura); }
    });
    // Cambio el contenido del textView por el id del Usuario
    Id_empleado.setText(idUsuario);

    // Dialogo AÑADIR
    MaterialAlertDialogBuilder builder = new MaterialAlertDialogBuilder(getContext());
    builder.setBackground(getContext().getResources().getDrawable(R.drawable.alert_dialog, theme: null));
    builder.setView(editLayout);
    // Si se pulsa Añadir
    builder.setPositiveButton(text: "Añadir", new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialogInterface, int i) {
            // Obtenemos los valores que ha introducido el usuario
            final String localizacion = Localizacion.getText().toString();
            final String m2_superficie = M2_superficie.getText().toString();
            final String malla_perforacion = Malla_perforacion.getText().toString();
            final String profundidad_barrenos = Profundidad_barrenos.getText().toString();
            final String numero_barrenos = Numero_barrenos.getText().toString();
            final String kg_explotivo = Kg_explotivo.getText().toString();
            final String precio = Precio.getText().toString();
            final String piedra_bruta = Piedra_bruta.getText().toString();
            final String fecha_voladura = Fecha_voladura.getText().toString();
            final String id_empleado = idUsuario;

            // Comprobamos que no estén vacios
            if(localizacion.isEmpty() || m2_superficie.isEmpty() || malla_perforacion.isEmpty() ||
                profundidad_barrenos.isEmpty() || numero_barrenos.isEmpty() || kg_explotivo.isEmpty() ||
                precio.isEmpty() || piedra_bruta.isEmpty() || fecha_voladura.isEmpty()) {
                Toast.makeText(getContext(), text: "Algunos campos están vacíos", Toast.LENGTH_SHORT).show();
            } else {
                // Hacemos una petición post para añadir la nueva voladura
                StringRequest stringRequest = new StringRequest(Request.Method.POST,Urls.ADD_VOLADURA_URL, new Response.Listener<String>() {
                    @Override

```

```

                    public void onResponse(String response) {
                        // Mensaje que tenemos desde el archivo php
                        Toast.makeText(getContext(), response, Toast.LENGTH_SHORT).show();
                        // Recargamos
                        Navigation.findNavController(view).navigate(R.id.action_voladurasListFragment_self);
                    }
                }, new Response.ErrorListener() {
                    @Override
                    public void onErrorResponse(VolleyError error) {
                        // Mensaje de error
                        Toast.makeText(getContext(), error.toString(), Toast.LENGTH_SHORT).show();
                    }
                });
            }{
                @Override
                protected Map<String, String> getParams() throws AuthFailureError {
                    // Creamos una instancia el objeto Map
                    HashMap<String, String> params = new HashMap<>();
                    // Con put ingresamos los datos a enviar
                    params.put("localizacion",localizacion);
                    params.put("m2_superficie",m2_superficie);
                    params.put("malla_perforacion",malla_perforacion);
                    params.put("profundidad_barrenos",profundidad_barrenos);
                    params.put("numero_barrenos",numero_barrenos);
                    params.put("kg_explotivo",kg_explotivo);
                    params.put("precio",precio);
                    params.put("piedra_bruta",piedra_bruta);
                    params.put("fecha_voladura",fecha_voladura);
                    params.put("id_empleado",id_empleado);

                    return params;
                }
            };
            // Creamos instancia de la clase RequestQueue
            RequestQueue requestQueue = Volley.newRequestQueue(getContext());
            // Agregamos la instancia de nuestro objeto StringRequest, ayuda a procesar las peticiones
        }
    }
}

```

```

    // Creamos instancia de la clase RequestQueue
    RequestQueue requestQueue = Volley.newRequestQueue(getContext());
    // Agregamos la instancia de nuestro objeto StringRequest, ayuda a procesar las peticiones
    requestQueue.add(stringRequest);
}

};

// Si se pulsa cancelar
builder.setNegativeButton(text: "Cancelar", new DialogInterface.OnClickListener() {
    @Override
    public void onClick(DialogInterface dialogInterface, int i) {
        // Cerramos el dialogo
        dialogInterface.dismiss();
    }
});
// Se muestra el dialogo
builder.show();

// Mostramos el dialogo de fecha y hora
private void showDateTimeDialog(EditText fecha_voladura) {
    final Calendar calendar = Calendar.getInstance();
    DatePickerDialog.OnDateSetListener dateSetListener = new DatePickerDialog.OnDateSetListener() {
        @Override
        public void onDateSet(DatePicker datePicker, int year, int month, int dayOfMonth) {
            calendar.set(Calendar.YEAR,year);
            calendar.set(Calendar.MONTH,month);
            calendar.set(Calendar.DAY_OF_MONTH,dayOfMonth);

            TimePickerDialog.OnTimeSetListener timeSetListener = new TimePickerDialog.OnTimeSetListener() {
                @Override
                public void onTimeSet(TimePicker timePicker, int hourOfDay, int minute) {
                    calendar.set(Calendar.HOUR_OF_DAY,hourOfDay);
                    calendar.set(Calendar.MINUTE,minute);
                }
            }
        }
    }
}

```

```

// Mostramos el dialogo de fecha y hora
private void showDateTimeDialog(EditText fecha_voladura) {
    final Calendar calendar = Calendar.getInstance();
    DatePickerDialog.OnDateSetListener dateSetListener = new DatePickerDialog.OnDateSetListener() {
        @Override
        public void onDateSet(DatePicker datePicker, int year, int month, int dayOfMonth) {
            calendar.set(Calendar.YEAR,year);
            calendar.set(Calendar.MONTH,month);
            calendar.set(Calendar.DAY_OF_MONTH,dayOfMonth);

            TimePickerDialog.OnTimeSetListener timeSetListener = new TimePickerDialog.OnTimeSetListener() {
                @Override
                public void onTimeSet(TimePicker timePicker, int hourOfDay, int minute) {
                    calendar.set(Calendar.HOUR_OF_DAY,hourOfDay);
                    calendar.set(Calendar.MINUTE,minute);

                    SimpleDateFormat simpleDateFormat = new SimpleDateFormat( pattern: "yyyy-MM-dd HH:mm");
                    fecha_voladura.setText(simpleDateFormat.format(calendar.getTime()));
                }
            };

            new TimePickerDialog(getContext(),timeSetListener,calendar.get(Calendar.HOUR_OF_DAY),calendar.get(Calendar.MINUTE), is24HourView: true).show();
        }
    };

    new DatePickerDialog(getContext(),dateSetListener,calendar.get(Calendar.YEAR),calendar.get(Calendar.MONTH),calendar.get(Calendar.DAY_OF_MONTH)).show();
}
}

```

Explicación de Métodos de la clase **VoladurasListFragment**.

- **onCreateView()** → Método que carga completamente la vista.
- **onViewCreate()** → Se llama inmediatamente después de **onCreateView** (el método que inicializa y crea todos sus objetos), la diferencia entre ambos métodos en que las asignaciones de subviews a los campos se deben de hacer en **onViewCreate()**.

Dentro de este método podemos encontrar que se hace referencia a **SharedPreferences** para poder acceder al id del Usuario que esta logueado.

Hacemos instancia al **RecyclerView**, este facilita que se muestren de manera eficiente grandes conjuntos de datos. Nosotros proporcionas los datos y definimos el aspecto de cada elemento, y la biblioteca **RecyclerView** creará los elementos de forma dinámica cuando se los necesite.

Este **RecyclerView** lo enlazaremos con un arraylist de voladuras, para ello llamamos al método **loadAllVoladuras()**.

También en **onCreateView()** comprobamos cuando el botón añadir ha sido pulsado, si esto es así llamará al método **addVoladuras()**.

- **loadAllVoladuras()** → Método encargado de que se muestren todas las voladuras existentes, para poder obtener todas, hacemos una consulta a la base de datos, ayudándonos con php, esta nos devolverá un JSONArray (clave/valor). Lo recorremos con un for para poder acceder a todas y cada una de las voladuras,

crearemos un objeto voladura y le asignaremos todos los valores obtenidos, después añadiremos esa voladura a el ArrayList que habíamos creado anteriormente.

Hacemos una instancia del objeto Adapter para que sea él quien gestiona el **RecyclerView**.

- **addVoladuras()** → Añade nuevas voladuras. Accedemos a los id de la vista para poder obtener los datos introducidos por el usuario, pinchando sobre el EditText de Fecha se abrirá una pequeña ventana que nos dará la opción de elegir la fecha y hora mucho más fácil.

Creamos un diálogo para no tener que estar navegando entre ventanas. Este se muestra sobre la lista de voladuras y es donde el usuario tendrá que introducir los datos necesarios para poder añadir una nueva voladura.

Comprobamos que los campos introducidos no estén vacíos y se lo enviamos al archivo .php mediante Map.

- **showDateTimeDialog(fecha)** → Mostrará la pequeña ventana de opción de fecha y hora.

3. Voladuras Adapter → Encargada mayoritariamente de eliminar, editar y mostrar los detalles de una voladura en concreto.

Un adaptador (**Adapter**) es un mecanismo de Android que hace de puente entre nuestros datos y las vistas contenidas en un ListView (o un GridView o Spinner).

ListView: Muestra una colección de vistas desplazables verticalmente, donde cada vista se coloca inmediatamente debajo de la vista anterior en la lista. Para un enfoque más moderno, flexible y eficaz para mostrar listas, nosotros usamos **RecyclerView**.



```
public class VoladurasAdapter extends RecyclerView.Adapter<VoladurasAdapter.VoladurasHolder> {  
    // Declaración de variables  
    Context context;  
    List<Voladuras> voladurasList;  
    // Para formatear los números decimales y enteros  
    NumberFormat nf = new DecimalFormat("###,##0");  
  
    // Constructor del Adapter  
    public VoladurasAdapter(Context context, List<Voladuras> voladurasList) {  
        this.context = context;  
        this.voladurasList = voladurasList;  
    }  
  
    @NonNull  
    @Override  
    public VoladurasHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {...}  
  
    @Override  
    public void onBindViewHolder(@NonNull VoladurasHolder holder, int position) {  
        Voladuras voladuras = voladurasList.get(position);  
        holder.id.setText(nf.format(voladuras.getId()));  
        holder.localizacion.setText(voladuras.getLocalizacion());  
        holder.m2_superficie.setText(nf.format(voladuras.getM2_superficie()));  
        holder.malla_perforacion.setText(voladuras.getMalla_perforacion());  
        holder.profundidad_barrenos.setText(nf.format(voladuras.getProfundidad_barrenos()));  
        holder.numero_barrenos.setText(nf.format(voladuras.getNumero_barrenos()));  
        holder.kg_explotivo.setText(nf.format(voladuras.getKg_explotivo()));  
        holder.precio.setText(nf.format(voladuras.getPrecio()));  
        holder.piedra_bruta.setText(nf.format(voladuras.getPiedra_bruta()));  
        holder.fecha_voladura.setText(voladuras.getFecha_voladura());  
        holder.id_empleado.setText(nf.format(voladuras.getId_empleado()));  
  
        Sharedpreferences preferences = context.getSharedPreferences("preferenciasLogin", Context.MODE_PRIVATE);  
    }  
}
```

```

String idUsuario = preferencecess.getString("idUsuario", "holo");
holder.btnMostrarVoladura.setOnClickListener(new View.OnClickListener() {
    @RequiresApi(api = Build.VERSION_CODES.LOLLIPOP)
    @Override
    public void onClick(View view) {
        // Instancia del archivo XML de diseño
        View editLayout = LayoutInflater.from(context).inflate(R.layout.voladuras_show, null);
        // Asignamos las id a las variables
        TextView Id = editLayout.findViewById(R.id.show_id);
        TextView Localizacion = editLayout.findViewById(R.id.show_localizacion);
        TextView M2_superficie = editLayout.findViewById(R.id.show_m2_superficie);
        TextView Malla_perforacion = editLayout.findViewById(R.id.show_malla_perforacion);
        TextView Profundidad_barrenos = editLayout.findViewById(R.id.show_profundidad_barrenos);
        TextView Numero_barreno = editLayout.findViewById(R.id.show_numero_barrenos);
        TextView Kg_explotivo = editLayout.findViewById(R.id.show_kg_explotivo);
        TextView Precio = editLayout.findViewById(R.id.show_precio);
        TextView Piedra_bruta = editLayout.findViewById(R.id.show_piedra_bruta);
        TextView Fecha_voladura = editLayout.findViewById(R.id.show_fecha_voladura);
        TextView Id_empleado = editLayout.findViewById(R.id.show_id_empleado);

        // Cambio el contenido del textView
        Id.setText(String.format(voladuras.getId()));
        Localizacion.setText(voladuras.getLocalizacion());
        M2_superficie.setText(String.format(voladuras.getM2_superficie()));
        Malla_perforacion.setText(voladuras.getMalla_perforacion());
        Profundidad_barrenos.setText(String.format(voladuras.getProfundidad_barrenos()));
        Numero_barrenos.setText(String.format(voladuras.getNumBarrenos()));
        Kg_explotivo.setText(String.format(voladuras.getKg_explotivo()));
        Precio.setText(String.format(voladuras.getPrecio()));
        Piedra_bruta.setText(String.format(voladuras.getPiedra_bruta()));
        Fecha_voladura.setText(voladuras.getFecha_voladura());
        Id_empleado.setText(idUsuario);

        Log.i("TAG", "ID USUARIO ADAPTER: " + idUsuario);
        Id_empleado.setText(idUsuario);
    }
});

MaterialAlertDialogBuilder builder = new MaterialAlertDialogBuilder(context);
builder.setBackground(context.getResources().getDrawable(R.drawable.alert_dialog, theme));
builder.setView(editLayout);
builder.setPositiveButton("Aceptar", new DialogInterface.OnClickListener() {
    @Override
    public void onClick(DialogInterface dialogInterface, int i) {
        dialogInterface.dismiss();
    }
});
builder.setNegativeButton("Eliminar", new DialogInterface.OnClickListener() {
    @Override
    public void onClick(DialogInterface dialogInterface, int i) {
        // Dialogo
        MaterialAlertDialogBuilder builder = new MaterialAlertDialogBuilder(context);
        builder.setBackground(context.getResources().getDrawable(R.drawable.alert_dialog, theme));
        builder.setTitle("Borrar Voladura?");
        builder.setMessage("Seguro que quieres borrar la voladura con la fecha: " + voladuras.getFecha_voladura() + "?");
        builder.setNegativeButton("Cancelar", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialogInterface, int i) {
                dialogInterface.dismiss();
            }
        });
        builder.setPositiveButton("Borrar", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialogInterface, int i) {
                StringRequest stringRequest = new StringRequest(Request.Method.POST,Urls.DELETE_VOLADURA_URL, new Response.Listener<String>() {
                    // Método que elimina una voladura en caso de que los datos obtenidos sean correctos
                    @Override
                    public void onResponse(String response) {
                        try {
                            JSONObject object = new JSONObject(response);
                            // Obtenemos el estado que nos devuelve el json
                            String check = object.getString("estado");
                            if (check.equals("ok")) {
                                // Si el estado es ok, borramos la posición del item
                                adapter.removeItem(position);
                            }
                        } catch (JSONException e) {
                            e.printStackTrace();
                        }
                    }
                });
                RequestQueue requestQueue = Volley.newRequestQueue(context);
                requestQueue.add(stringRequest);
            }
        });
    }
});

```

Explicación de Métodos de la clase **VoladurasAdapter**.

- **VoladurasAdapter()** → Constructor del Adapter.
- **onCreateViewHolder()** → Se llama cuando RecyclerView necesita un nuevo RecyclerView.ViewHolder (**VoladurasHolder**) del tipo dado para representar un elemento.

Este nuevo ViewHolder debe construirse con una nueva Vista que pueda representar los elementos del tipo dado. Puede crear una nueva vista manualmente o inflarla desde un archivo de diseño XML.(Nosotros lo inflamos)

El nuevo ViewHolder se usará para mostrar elementos del adaptador que usa **onBindViewHolder()**.

Un ViewHolder describe una vista de elemento y metadatos sobre su lugar dentro de RecyclerView.

- **onBindViewHolder()** → Los llama RecyclerView para mostrar los datos en la posición específica. Este método actualiza el contenido del ViewHolder.itemView para reflejar el elemento en la posición dada.

Hay dos botones en cada voladura, detalles y editar, ambos nos muestran un diálogo con los detalles de la voladura seleccionada, el botón detalles nos muestra todos los datos de la voladura y nos dará la opción de poder eliminar, y en la opción de editar podremos cambiar los datos de la voladura.

- **getItemCount()** → Devuelve el tamaño de la lista.
- **borrarVoladura(item)** → Borramos la voladura seleccionada.
- **showDateTimeDialog(fecha)** → Como hemos dicho anteriormente, mostrará una pequeña ventana de opción de fecha y hora.

INFORMACIÓN :

A partir de aquí la mayoría de las clases hacen lo mismo que las tres que acabo de explicar, por lo que para no repetir contenido mostraré algunas pequeñas variaciones.

Las clases que utilizan el mismo código que voladuras son, productos, consumos y stocks . Una pequeña diferencia que encontramos en productos es cuando vayamos a añadir.

(Método addProductos)



```
FECHA_VOLADURA.setInputType(InputType.TYPE_NULL);

LoadAllVoladuras(FECHA_VOLADURA);

FECHA_VOLADURA.addTextChangedListener(new TextWatcher() {
    @Override
    public void beforeTextChanged(CharSequence charSequence, int i, int i1, int i2) {}

    @Override
    public void onTextChanged(CharSequence charSequence, int i, int i1, int i2) {}

    @Override
    public void afterTextChanged(Editable editable) {
        if(FECHA_VOLADURA.requestFocus()) {
            FECHA_VOLADURA.setInputType(InputType.TYPE_NULL);
        }
        fechaSpinner = FECHA_VOLADURA.getText().toString();
        Log.i("tag: " + "La fecha elegida es:", fechaSpinner);
        String newText = FECHA_VOLADURA.getText().toString();
        Log.i("tag: " + "Hola", newText);
    }
});

fechaSpinner = FECHA_VOLADURA.getText().toString();
ID_EMPLEADO.setText(ID_USUARIO);

MaterialAlertDialogBuilder builder = new MaterialAlertDialogBuilder(getContext());
builder.setBackground(getContext().getResources().getDrawable(R.drawable.alert_dialog, null));
builder.setView(editLayout);
```

```

// Añadimos a un arraylist todas las fechas de las voladuras existentes
private void cargarFechasVoladura(){
    JsonArrayRequest request = new JsonArrayRequest(Urls.SHOW_ALL_VOLADURA_DATA_URL, new Response.Listener<JSONArray>() {
        @Override
        public void onResponse(JSONArray response) {
            try {
                for(int i = 0; i < response.length(); i++){
                    JSONObject object = response.getJSONObject(i);
                    String voladura_fecha = object.getString( name: "fecha_voladura").trim();

                    /*Log.i("fecha voladura: ",voladura_fecha);*/

                    fechaList.add(voladura_fecha);
                }

                fechaAdapter = new ArrayAdapter<>(getContext(),R.layout.option_item,fechaList);

            } catch (JSONException e) {
                e.printStackTrace();
            }
        }
    }, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {
            Toast.makeText(getContext(), error.toString(), Toast.LENGTH_SHORT).show();
        }
    });
    RequestQueue queue = Volley.newRequestQueue(getContext());
}

```

```

recyclerView.setLayoutManager(new LinearLayoutManager(getContext()));

productosList = new ArrayList<>();

LoadAllProductos();

}

// Cargamos todos los productos en el recyclerView
private void LoadAllProductos() {...}

// Añadimos un nuevo producto
@RequiresApi(api = Build.VERSION_CODES.LOLLIPOP)
private void AddProducto(View view) {...}

// Añadimos a un arraylist todas las fechas de las voladuras existentes
private void cargarFechasVoladura(){...}

// Cargamos el arraylist en el dropdown
private void loadAllVoladuras(AutoCompleteTextView dropdown) {
    dropdown.setText(fechaAdapter.getItem( position: 0).toString(), filter: false);
    dropdown.setAdapter(fechaAdapter);
}
}

```

Explicación de Métodos de la clase **Productos**.

- **cargarFechasVoladura()** → Nos devuelve un arrayList con todas las fechas de las voladuras existentes. (Puesto que de una voladura se corresponden sus productos)
- **loadAllVoladuras()** → Cargamos el arrayList en el dropdown.
- **addProductos()** → Es igual que addVoladuras(). Sólo que cuando vayamos a elegir una fecha nos aparecerá el diálogo con un dropdown con las fechas de todas la voladuras. Sabemos cuando elegimos o cambiamos la fecha con **addTextChangedListener()**.

5.- MenuProductosFragment / MenuConsumosFragment → Clase que nos redirige a productos / consumos o a su stock. Mostraré uno de ellos ya que el código es exactamente igual.

```
public class MenuProductosFragment extends Fragment {
    private CardView cvProductos, cvStockProductos;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                           Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.fragment_menu_productos, container, false);
    }

    @Override
    public void onViewCreated(@NonNull View view, Bundle savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);

        cvProductos = (CardView) view.findViewById(R.id.cvProductos);
        cvStockProductos = (CardView) view.findViewById(R.id.cvStockProductos);

        final NavController navController = Navigation.findNavController(view);

        // Botón Products
        cvProductos.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                navController.navigate(R.id.action_menuProductosFragment_to_productosListFragment);
            }
        });

        // Botón Stock Products
        cvStockProductos.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                navController.navigate(R.id.action_menuProductosFragment_to_stockProductosListFragment);
                //Log.i("Entra", "Botón Products");
            }
        });
    }
}
```

Explicación de código de la clase **MenuProductosFragment / MenuConsumos Fragment**.

En la vista de estos fragment tenemos dos CardView que utilizamos como botones.

Estas CardView son una implementación que nos proporciona Google del elemento visual en forma de tarjetas de información.

Cuando pulsamos una de estas cardview, nos redirigirá a otro fragment correspondiente a la cardview pulsada. Para la navegación entre fragment hacemos uso del objeto Navigation.

6.- Control Horario / Registro_Entradas_Salidas → Clase que realiza el check in y check out.

```
public class Registro_Entradas_Salidas extends Fragment {
    CardView cvCheckIn, cvCheckOut;
    TextView tvCheckInL, tvCheckOutL, tvCheckInS, tvCheckOutS;
    RelativeLayout rlCheckIn, rlCheckOut;
    private String idusuario, msgRegistro;
    TextView tvRegistro;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                           Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.fragment_registro_entradas_salidas, container, false);
    }

    @Override
    public void onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);

        SharedPreferences preferencesss = getContext().getSharedPreferences("preferenciasLogin", Context.MODE_PRIVATE);
        // Recogemos los datos de las preferencias
        idusuario = preferencesss.getString("idUsuario", "holo");
        msgRegistro = preferencesss.getString("nombreUser", "nombreUsuario");

        // Id del view
        tvRegistro = (TextView) view.findViewById(R.id.tvRegistro);
        cvCheckIn = (CardView) view.findViewById(R.id.cvCheckIn);
        cvCheckOut = (CardView) view.findViewById(R.id.cvCheckOut);
        rlCheckIn = (RelativeLayout) view.findViewById(R.id.checkInRelative);
        rlCheckOut = (RelativeLayout) view.findViewById(R.id.checkOutRelative);
        tvCheckInL = (TextView) view.findViewById(R.id.t3);
        tvCheckOutL = (TextView) view.findViewById(R.id.t4);
        tvCheckInS = (TextView) view.findViewById(R.id.tvCheckIn);
        tvCheckOutS = (TextView) view.findViewById(R.id.tvCheckOut);
    }
}
```

```

cvCheckOut.setEnabled(false);
rlCheckOut.setBackgroundColor(Color.GRAY);

// Cambiar tonos de color
tvCheckOutL.setTextColor(getResources().getColor(R.color.mensaje));
tvCheckOutS.setTextColor(getResources().getColor(R.color.mensaje));

// Mensaje de bienvenida
tvRegistro.setText("Hola "+msgRegistro+"!");

// Consulta a la base de datos
StringRequest stringRequest = new StringRequest(Request.Method.POST,Urls.REGISTRO_DATA_URL, new Response.Listener<String>() {
    @Override
    public void onResponse(String response) {
        try {
            JSONObject object = new JSONObject(response);

            String check = object.getString( name: "estado");

            // Cambiamos la tonalidad de los botones si el usuario ha realizado el check in o check out
            switch (check){
                case "inicio":
                    cvCheckOut.setEnabled(false);
                    rlCheckOut.setBackgroundColor(Color.GRAY);
                    tvCheckOutL.setTextColor(getResources().getColor(R.color.mensaje));
                    tvCheckOutS.setTextColor(getResources().getColor(R.color.mensaje));
                    break;
                case "checkout":
                    cvCheckOut.setEnabled(false);
                    rlCheckOut.setBackgroundColor(Color.GRAY);
                    tvCheckOutL.setTextColor(getResources().getColor(R.color.mensaje));
                    tvCheckOutS.setTextColor(getResources().getColor(R.color.mensaje));
                    cvCheckIn.setEnabled(false);
                    rlCheckIn.setBackgroundColor(Color.GRAY);
                    tvCheckInL.setTextColor(getResources().getColor(R.color.mensaje));
                    break;
            }
        } catch (JSONException e) {
            e.printStackTrace();
        }
    }
}, new Response.ErrorListener() {
    @Override
    public void onErrorResponse(VolleyError error) {
        Toast.makeText(getApplicationContext(), error.toString(), Toast.LENGTH_SHORT).show();
    }
});

@Override
protected Map<String, String> getParams() throws AuthFailureError {
    HashMap<String, String> params = new HashMap<>();
    params.put("id_empleado", idUsuario);

    return params;
}
};

RequestQueue requestQueue = Volley.newRequestQueue(getApplicationContext());

```

```

tvCheckInS.setTextColor(getResources().getColor(R.color.mensaje));
Snackbar.make(view, text: "Tu registro está completo por hoy:", Snackbar.LENGTH_LONG)
    .setAction( text: "Action", listener: null).show();
break;
case "checkin":
    cvCheckOut.setEnabled(true);
    rlCheckOut.setBackgroundColor(Color.WHITE);
    cvCheckIn.setEnabled(false);
    rlCheckIn.setBackgroundColor(Color.GRAY);
    tvCheckInL.setTextColor(getResources().getColor(R.color.mensaje));
    tvCheckInS.setTextColor(getResources().getColor(R.color.mensaje));
    break;
}

} catch (JSONException e) {
    e.printStackTrace();
}

},
new Response.ErrorListener() {
    @Override
    public void onErrorResponse(VolleyError error) {
        Toast.makeText(getApplicationContext(), error.toString(), Toast.LENGTH_SHORT).show();
    }
});

@Override
protected Map<String, String> getParams() throws AuthFailureError {
    HashMap<String, String> params = new HashMap<>();
    params.put("id_empleado", idUsuario);

    return params;
}
};

RequestQueue requestQueue = Volley.newRequestQueue(getApplicationContext());

```

```

RequestQueue requestQueue = Volley.newRequestQueue(getApplicationContext());
requestQueue.add(stringRequest);

// Se ha realizado el Check In, por lo que añadimos el registro a la base de datos
cvCheckIn.setOnClickListener(new View.OnClickListener() {
    @RequiresApi(api = Build.VERSION_CODES.LOLLIPOP)
    @Override
    public void onClick(View view) {

        // Obtenemos la fecha y hora del dia
        Date fechaEntrada = new Date();
        DateFormat horaDateFormat = new SimpleDateFormat( pattern: "yyyy-MM-dd HH:mm:ss");

        // Dialogo
        MaterialAlertDialogBuilder builder = new MaterialAlertDialogBuilder(getApplicationContext());
        builder.setBackground(getApplicationContext().getResources().getDrawable(R.drawable.alert_dialog, theme: null));
        builder.setTitle("Realizar Check In?");
        builder.setMessage("Registrar la Entrada en esta fecha y hora?"+horaDateFormat.format(fechaEntrada));
        builder.setPositiveButton( text: "Guardar", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialogInterface, int i) {

                StringRequest stringRequest = new StringRequest(Request.Method.POST,Urls.ADD_REGISTRO_DATA_URL, new Response.Listener<String>() {
                    @Override
                    public void onResponse(String response) {
                        try {
                            JSONObject object = new JSONObject(response);

                            String check = object.getString( name: "estado");

                            if(check.equals("checkin")){
                                Snackbar.make(view, text: "Registro de entrada realizado!", Snackbar.LENGTH_LONG)
                                    .setAction( text: "Action", listener: null).show();
                                cvCheckOut.setEnabled(true);
                                rlCheckOut.setBackgroundColor(Color.WHITE);
                            }
                        } catch (JSONException e) {
                            e.printStackTrace();
                        }
                    }
                });
            }
        });
    }
});

```

```

        tvCheckOutL.setTextColor(Color.BLACK);
        tvCheckOutS.setTextColor(Color.BLACK);
        cvCheckIn.setEnabled(false);
        rlCheckIn.setBackgroundDrawable(Color.GRAY);
        tvCheckInL.setTextColor(getResources().getColor(R.color.mensaje));
        tvCheckInS.setTextColor(getResources().getColor(R.color.mensaje));
    }

} catch (JSONException e) {
    e.printStackTrace();
}
}

}, new Response.ErrorListener() {
    @Override
    public void onErrorResponse(VolleyError error) {
        Toast.makeText(getApplicationContext(), error.toString(), Toast.LENGTH_SHORT).show();
    }
});

@Override
protected Map<String, String> getParams() throws AuthFailureError {
    HashMap<String, String> params = new HashMap<>();
    params.put("fecha_check_in", hourdateFormat.format(fechaEntrada));
    params.put("id_empleado", idUsuario);

    return params;
};

RequestQueue requestQueue = Volley.newRequestQueue(getApplicationContext());
requestQueue.add(stringRequest);
}
});
builder.setNegativeButton("Cancelar", new DialogInterface.OnClickListener() {
    @Override

```

```

    builder.setNegativeButton("Cancelar", new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialogInterface, int i) {
            dialogInterface.dismiss();
        }
    });
    builder.show();
}
});

cvCheckOut.setOnClickListener(new View.OnClickListener() {
    @RequiresApi(api = Build.VERSION_CODES.LOLLIPOP)
    @Override
    public void onClick(View view) {

        Date fechaSalida = new Date();
        DateFormat hourdateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

        MaterialAlertDialogBuilder builder = new MaterialAlertDialogBuilder(getApplicationContext());
        builder.setBackground(getApplicationContext().getResources().getDrawable(R.drawable.alert_dialog, null));
        builder.setTitle("Realizar Check Out?");
        builder.setMessage("Registro de Salida en esta fecha y hora " + hourdateFormat.format(fechaSalida));
        builder.setPositiveButton("Guardar", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialogInterface, int i) {

StringRequest stringRequest = new StringRequest(Request.Method.POST,Urls.UPDATE_REGISTRO_DATA_URL, new Response.Listener<String>() {
    @Override
    public void onResponse(String response) {
        try {
            JSONObject object = new JSONObject(response);

            String check = object.getString("estado");


```

```

                if(check.equals("checkout")){
                    Snackbar.make(view, "¡Registro de salida realizado!", Snackbar.LENGTH_LONG)
                        .setAction("Action", null).show();
                    cvCheckOut.setEnabled(false);
                    rlCheckOut.setBackgroundDrawable(Color.GRAY);
                    tvCheckOutL.setTextColor(getResources().getColor(R.color.mensaje));
                    tvCheckOutS.setTextColor(getResources().getColor(R.color.mensaje));
                }

            } catch (JSONException e) {
                e.printStackTrace();
            }
        }, new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error) {
                Toast.makeText(getApplicationContext(), error.toString(), Toast.LENGTH_SHORT).show();
            }
        });

        @Override
        protected Map<String, String> getParams() throws AuthFailureError {
            HashMap<String, String> params = new HashMap<>();
            params.put("fecha_check_out", hourdateFormat.format(fechaSalida));
            params.put("id_empleado", idUsuario);

            return params;
        };

        RequestQueue requestQueue = Volley.newRequestQueue(getApplicationContext());
        requestQueue.add(stringRequest);
    }
});
builder.setNegativeButton("Cancelar", new DialogInterface.OnClickListener() {
    @Override

```

Explicación de Métodos de la clase Registro_Entradas_Salidas.

En esta clase hacemos el check in (Cuando empieza la jornada de trabajo) y check out (cuando termina la jornada de trabajo)

- **onViewCreated()** → Cuando recién entra el usuario a la app, el botón check in estará de color blanco y el de check out en gris. Esto que significa, que el único botón que funciona en ese momento es check in, mientras check out estará setEnable(false), o sea, que aunque se pinche en él no hará nada.

Además dentro de este, hacemos una consulta para saber si el usuario ha realizado el check in o el chek out, para ir orientando al usuario sobre los colores de los botones, sobre cual puede pulsar o cual no.

Cuando los botones se pulsen, estos actualizarán el registro de entrada y salida del día.

7.- Configuración → Clase encargada de mostrar los datos del usuario y donde poder cambiar la contraseña.

```
public class ConfiguracionFragment extends Fragment {  
    private String idUsuario;  
    private String nombreUsuario;  
    private String emailUsuario;  
    TextView tvNombre, tvEmail;  
    Button btnPass;  
  
    @Override  
    public View onCreateView(LayoutInflater inflater, ViewGroup container,  
                             Bundle savedInstanceState) {  
        // Inflate the layout for this fragment  
        return inflater.inflate(R.layout.fragment_configuracion, container, false);  
    }  
  
    @Override  
    public void onViewCreated(@NotNull @Nullable View view, @Nullable @org.jetbrains.annotations.Nullable Bundle savedInstanceState) {  
        super.onViewCreated(view, savedInstanceState);  
  
        SharedPreferences preferences = getContext().getSharedPreferences("preferenciasLogin", Context.MODE_PRIVATE);  
        // Recoger datos de las preferencias  
        idUsuario = preferences.getString("idUsuario", "1");  
        nombreUsuario = preferences.getString("nombreUser", "nombre");  
        emailUsuario = preferences.getString("email", "email");  
  
        tvNombre = (TextView) view.findViewById(R.id.nombre_usuario);  
        tvEmail = (TextView) view.findViewById(R.id.email_usuario);  
        btnPass = (Button) view.findViewById(R.id.change_password);  
  
        // Cambiar texto de los textview por los datos obtenidos de dichas preferencias  
        tvNombre.setText(nombreUsuario);  
        tvEmail.setText(emailUsuario);  
  
        // Botón cambiar contraseña  
        btnPass.setOnClickListener(new View.OnClickListener() {  
  
            // Botón cambiar contraseña  
            btnPass.setOnClickListener(new View.OnClickListener() {  
                @Override  
                public void onClick(View view) {changePassword(view);}  
            });  
        }));  
  
        // Método que cambia la contraseña del usuario  
        private void changePassword(View view){  
            StringRequest stringRequest = new StringRequest(Request.Method.POST,Urls.LOAD_CONFIGURATION_DATA_URL, new Response.Listener<String>(){  
                @RequiresApi(api = Build.VERSION_CODES.LOLLIPOP)  
                @Override  
                public void onResponse(String response) {  
                    View addLayout = LayoutInflater.from(getContext()).inflate(R.layout.change_pass, null);  
                    // Obtenir id de la vista  
                    EditText etActuPass = addLayout.findViewById(R.id.actual_pass);  
                    EditText etNewPass = addLayout.findViewById(R.id.new_pass);  
                    EditText etVeryPass = addLayout.findViewById(R.id.verificar_pass);  
  
                    // Dialogo  
                    MaterialAlertDialogBuilder builder = new MaterialAlertDialogBuilder(getContext());  
                    builder.setBackground(getContext().getResources().getDrawable(R.drawable.alert_dialog, theme null));  
                    builder.setTitle("¿Quieres cambiar tu contraseña?");  
                    builder.setView(addLayout);  
                    builder.setPositiveButton("Añadir", new DialogInterface.OnClickListener() {  
                        @Override  
                        public void onClick(DialogInterface dialogInterface, int i) {  
                            final String actual_pass = etActuPass.getText().toString();  
                            final String new_pass = etNewPass.getText().toString();  
                            final String verificar_pass = etVeryPass.getText().toString();  
                            final String nombre_user = nombreUsuario;  
                            final String email_user = emailUsuario;  
                            final String id_empleado = idUsuario;  
  
                            // Comprobar si los campos están vacíos  
                        }  
                    });  
                    builder.show();  
                }  
            });  
            RequestQueue requestQueue = Volley.newRequestQueue(getContext());  
            requestQueue.add(stringRequest);  
        }  
    }  
}
```

```

        // Comprobar si los campos están vacíos
        if(actual_pass.isEmpty() || new_pass.isEmpty() || verificar_pass.isEmpty()){
            Toast.makeText(getApplicationContext(), "Los campos no pueden estar vacíos", Toast.LENGTH_SHORT).show();
        } else if(!new_pass.equals(verificar_pass)){
            Toast.makeText(getApplicationContext(), "La contraseña verificada no es igual a la nueva.", Toast.LENGTH_SHORT).show();
        } else {
            // Consulta a la base de datos
            StringRequest stringRequest = new StringRequest(Request.Method.POST,Urls.LOAD_CONFIGURATION_DATA_URL, new Response.Listener<String>() {
                @Override
                public void onResponse(String response) {
                    Toast.makeText(getApplicationContext(), response, Toast.LENGTH_SHORT).show();
                    //Recargamos
                    Navigation.findNavController(view).navigate(R.id.action_configuracionFragment_self);
                }
            }, new Response.ErrorListener() {
                @Override
                public void onErrorResponse(VolleyError error) {
                    Toast.makeText(getApplicationContext(), error.toString(), Toast.LENGTH_SHORT).show();
                }
            }){
                @Override
                protected Map<String, String> getParams() throws AuthFailureError {
                    HashMap<String, String> params = new HashMap<>();
                    // Enviamos datos
                    params.put("nombre", nombre_user);
                    params.put("email", email_user);
                    params.put("password", actual_pass);
                    params.put("new_password", new_pass);
                    params.put("verify_password", verificar_pass);
                    params.put("id_empleado", id_empleado);

                    return params;
                }
            };
        }
    }
}

```

```

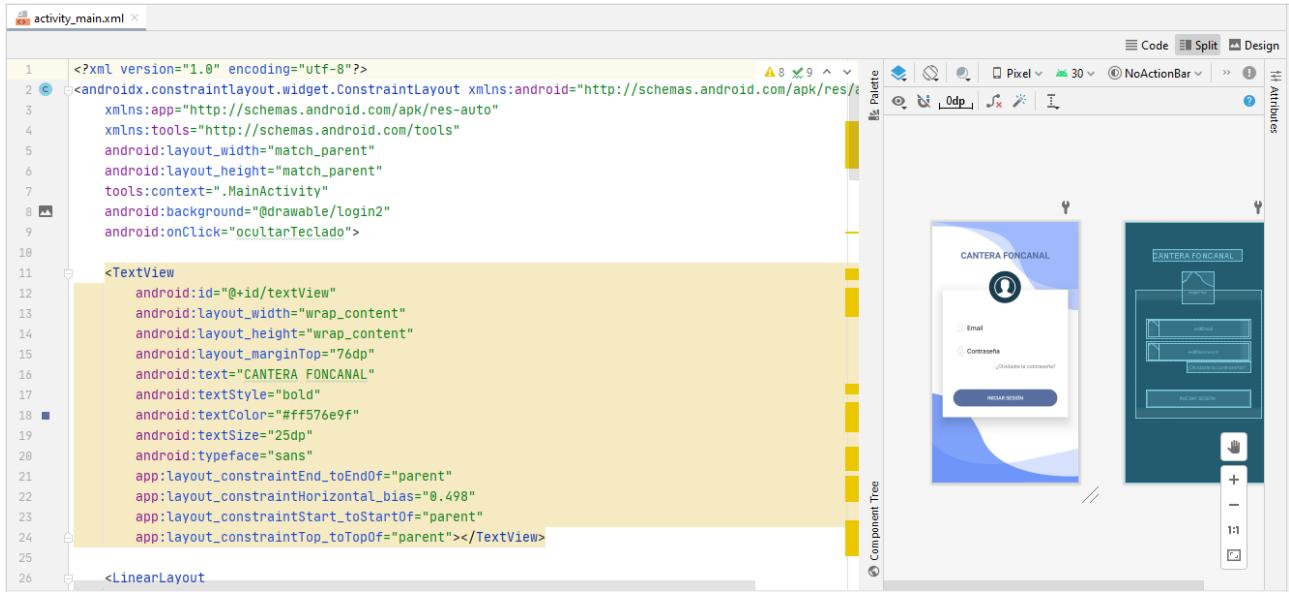
    });
    RequestQueue requestQueue = Volley.newRequestQueue(getApplicationContext());
    requestQueue.add(stringRequest);
}
});
builder.setNegativeButton("Cancelar", new DialogInterface.OnClickListener() {
    @Override
    public void onClick(DialogInterface dialogInterface, int i) {
        dialogInterface.dismiss();
    }
});
builder.show();
}, new Response.ErrorListener() {
    @Override
    public void onErrorResponse(VolleyError error) {
        Toast.makeText(getApplicationContext(), error.getMessage(), Toast.LENGTH_SHORT).show();
    }
});
// Instancia objeto
RequestQueue requestQueue = Volley.newRequestQueue(getApplicationContext());
// Agregamos la instancia de nuestro objeto StringRequest, ayuda a procesar las peticiones
requestQueue.add(stringRequest);
}
}

```

Explicación del código de la clase **Configuración**.

En esta clase como operación sólo podremos cambiar la contraseña. Mostramos un diálogo al usuario con tres campo vacíos: en el primero deberá de introducir su contraseña actual, en el segundo su nueva contraseña y en le tercero verificarla. Hacemos una consulta a la base de datos para comprobar que los datos son correctos, si lo son, la nueva contraseña se actualizará y si no, se notificará al usuario de que los datos no han sido correctos.

8.- .xml → Los diseños de android son archivos .xml que contienen views (botones, campos de texto....) con los que el usuario interactúa.



Explicación código .xml

Atributos

Cada objeto View admite su propia variedad de atributos XML. Algunos atributos son específicos de un objeto View (por ejemplo, TextView admite el atributo **textSize**), aunque estos atributos también son heredados por cualquier objeto View que pueda extender esta clase. Algunos son comunes para todos los objetos View, porque se heredan de la clase raíz View (como el atributo **id**). Además, otros atributos se consideran "parámetros de diseño", que son atributos que describen ciertas orientaciones de diseño de View.

ID

Cualquier objeto View puede tener un ID entero asociado para identificarse de forma única dentro del árbol. Cuando se compila la aplicación, se hace referencia a este ID como un número entero, pero normalmente se asigna el ID en el archivo XML de diseño como una string del atributo **id**. Este es un atributo XML común para todos los objetos View (definido por la clase **View**) y lo utilizarás muy a menudo.

Conexiones / Consultas a la base de datos Android-PHP

- **conexion.php** → Creamos la conexión con la base de datos.

```
❶ conexion.php X
❷ conexion.php > ...
1  <?php
2      $sql_host="localhost";
3      $sql_usuario="root";
4      $sql_pass="";
5      $sql_db="cantera";
6
7      $bd = new mysqli($sql_host, $sql_usuario, $sql_pass, $sql_db);
8      if ($bd->connect_error) { die('Error de Conexión ('.$mysqli->connect_errno.') '.$mysqli->connect_error); }
9      mysqli_set_charset($bd,"utf8");
10
11 ?>
```

- **validar_usuario.php** → Pertenece al login, comprobamos si el usuario existe y si la contraseña es correcta.

```
❶ validar_usuario.php > ...
1  <?php
2  // Abrimos conexión con la base de datos
3  require 'conexion.php';
4
5  // Obtenemos los valores por post
6  $usu_email = $_POST['email'];
7  $usu_password = $_POST['password'];
8
9  // Nos ayuda a prevenir ataques de inyecciones sql
10 $consulta = $bd->prepare("SELECT * FROM users WHERE email=?");
11 $consulta->bind_param('s',$usu_email);
12 $consulta->execute();
13
14 $resultado = $consulta->get_result();
15 if($fila = $resultado->fetch_assoc()){
16     // Verificamos la contraseña
17     if(password_verify($usu_password,$fila['password'])){
18         echo json_encode($fila,JSON_UNESCAPED_UNICODE);
19     }
20 }
21
22 // Cerramos conexión
23 $consulta->close();
24 $bd->close();
25
26 ?>
```

- **voladuras / voladuras.php** → Devuelve todas la voladuras.

```
(voladuras) voladuras.php X
voladuras > (voladuras) voladuras.php > ...
1 3?php
2
3 //Conexión a la base de datos
4 require '../conexion.php';
5
6 //Consulta que devuelve todas las voladuras ordenadas por fecha
7 $consulta = $bd->query("SELECT * FROM voladuras ORDER BY fecha_voladura DESC");
8 $voladuras = array();
9
10 if(!$consulta){
11     printf("Error: %s\n", $bd->error);
12     exit();
13 } else {
14     // Añadimos el resultado de la consulta a un array
15     while($row = $consulta->fetch_assoc()){
16
17         $voladuras[] = $row;
18     }
19 }
20 echo json_encode($voladuras,JSON_UNESCAPED_UNICODE);
21
22 ?>
```

- **voladuras / add_voladuras.php** → Añade una nueva voladura.

```
(voladuras) add_voladura.php X
voladuras > (voladuras) add_voladura.php > ...
1 3?php
2 // Conexión a la base de datos
3 require '../conexion.php';
4 require '../funciones/funciones.php';
5
6 // valores obtenidos por post
7 $localizacion = $_POST['localizacion'];
8 $m2_superficie = $_POST['m2_superficie'];
9 $malla_perforacion = $_POST['malla_perforacion'];
10 $profundidad_barrenos = $_POST['profundidad_barrenos'];
11 $numero_barrenos = $_POST['numero_barrenos'];
12 $kg_explotivo = $_POST['kg_explotivo'];
13 $precio = $_POST['precio'];
14 $piedra_bruta = $_POST['piedra_bruta'];
15 $id_empleado = intval($_POST['id_empleado']);
16 $fecha_voladura = date('Y-m-d H:i:s',strtotime($_POST['fecha_voladura']));
17
18 // Comprobamos si algun valor es incorrecto
19 if(comprobarNumero($m2_superficie) && comprobarNumero($profundidad_barrenos) && comprobarNumero($numero_barrenos) &&
20     comprobarNumero($kg_explotivo) && comprobarNumero($precio) && comprobarNumero($piedra_bruta)){
21
22     // Insertamos una nueva voladura
23     $consulta = $bd->query("INSERT INTO voladuras (localizacion,m2_superficie,malla_perforacion,profundidad_barrenos,numero_barrenos,kg_explotivo,precio,piedra_bruta,id_empleado,fecha_voladura) VALUES ('{$localizacion}',{$m2_superficie},{$malla_perforacion},{$profundidad_barrenos},{$numero_barrenos},{$kg_explotivo},{$precio}, '{$piedra_bruta}',{$id_empleado},'{$_POST['fecha_voladura']}' )");
24
25     if(!$consulta){
26         printf("Error: %s\n", $bd->error);
27         exit();
28     } else {
29         echo 'Voladura insertada con éxito';
30     }
31 }
32 } else {
```

- **voladuras / editar_voladura.php** → Actualizamos la voladura seleccionada.

```
editar_voladura.php X
voladuras > editar_voladura.php > ...
1  <?php
2  // Conexión a la base de datos
3  require '../conexion.php';
4  require '../funciones/funciones.php';
5
6  // Obtenemos valores por post
7  $id = intval($_POST['id']);
8  $localizacion = $_POST['localizacion'];
9  $m2_superficie = $_POST['m2_superficie'];
10 $mall_perforacion = $_POST['mall_perforacion'];
11 $profundidad_barrenos = $_POST['profundidad_barrenos'];
12 $numero_barrenos = $_POST['numero_barrenos'];
13 $kg_explotivo = $_POST['kg_explotivo'];
14 $precio = $_POST['precio'];
15 $piedra_bruta = $_POST['piedra_bruta'];
16 $id_empleado = intval($_POST['id_empleado']);
17 $fecha_voladura = date('Y-m-d H:i:s', strtotime($_POST['fecha_voladura']));
18
19 // Comprobamos si algun valor es incorrecto
20 if(comprobarNumero($m2_superficie) && comprobarNumero($profundidad_barrenos) && comprobarNumero($numero_barrenos) &&
21     comprobarNumero($kg_explotivo) && comprobarNumero($precio) && comprobarNumero($piedra_bruta)){
22
23     // Actualizamos la voladura
24     $consulta = $bd->query("UPDATE voladuras SET localizacion = '$localizacion', m2_superficie = '$m2_superficie', mall_
25     profundidad_barrenos = '$profundidad_barrenos', numero_barrenos = '$numero_barrenos', kg_explotivo = '$kg_explotivo',
26     fecha_voladura = '$fecha_voladura', id_empleado=' $id_empleado' WHERE id='$id'");
27
28     if(!$consulta){
29         printf("Error: %s\n", $bd->error);
30         exit();
31     } else {
32         echo 'Actualizado con éxito';
33     }
34 }
```

- **voladuras / borrar_voladura.php** → Eliminar la voladura seleccionada.

```
borrar_voladura.php X
voladuras > borrar_voladura.php > ...
1  <?php
2  // Conexión a la base de datos
3  require '../conexion.php';
4  require '../funciones/funciones.php';
5
6  $id = intval($_POST['id']);
7  $resultado = array();
8
9  // Consultamos si existe la voladura
10 $consulta = $bd->query("SELECT * FROM voladuras WHERE id=$id");
11
12 if(!$consulta){
13     printf("Error: %s\n", $bd->error);
14     exit();
15 } else [
16     // Eliminamos la voladura
17     $consulta2 = $bd->query("DELETE FROM voladuras WHERE id=$id");
18     if(!$consulta2){
19         printf("Error: %s\n", $bd->error);
20         exit();
21     } else {
22         $resultado['estado'] = "borrar";
23         echo json_encode($resultado,JSON_UNESCAPED_UNICODE);
24     }
25 ]
26
27
28 ?>
```

- **funciones / funciones.php** → Recoge las funciones que son necesarias durante la ejecución. Ahora mismo hay una, ya que sólo nos hace falta comprobar que los números estén bien escritos.

```
funciones.php X
funciones > funciones.php > ...
1  <?php
2
3  //Validar número real / entero
4  function comprobarNumero($num)
5  {
6      return preg_match("/^(\d{1,10})(\.\d{1,3})?$/", $num);
7  }
8
9
10
11 ?>
```

- **registro / registro.php** → Consulta el estado del control horario del usuario.

```
registro.php X
registro > registro.php > ...
1  <?php
2  // Conexion a la base de datos
3  require '../conexion.php';
4  require '../funciones/funciones.php';
5
6  // Establecemos la zona horaria
7  date_default_timezone_set('Europe/Madrid');
8
9  //echo date('l jS \of F Y H:i:s A');
10 //echo date('d/m/Y H:i:s');
11 //echo date('Y-m-d');
12
13 // Obtenemos la fecha de hoy
14 $fechaLocal = strtotime(date('Y-m-d'));
15
16 $resultado = array();
17
18 $fechaEntrada = "";
19 $fechaSalida = "";
20
21 // Valores obtenidos por post
22 $fechaEntrada = isset($_POST['fecha_check_in'])? intval($_POST['fecha_check_in']): '';
23 $fechaSalida = isset($_POST['fecha_check_out'])? intval($_POST['fecha_check_out']): '';
24 $id_empleado = intval($_POST['id_empleado']);
25
26 // Consulta para saber si el usuario ya ha realizado el check in
27 $consultaE = $bd->query("SELECT * FROM presencialidades WHERE id_empleado=$id_empleado AND fecha='$fechaLocal'");
28
29 // Devolveremos un estado según si el usuario ha realizado el check in o check out
30 if (mysqli_num_rows($consultaE) > 0) {
31
32     if($row = $consultaE->fetch_assoc()){
33         $fechaSalida = $row['fecha_check_out'];
34     }
35 }
```

```

// Devolveremos un estado según si el usuario ha realizado el check in o check out
if (mysqli_num_rows($consultaE) > 0) {

    if($row = $consultaE->fetch_assoc()){
        $fechaSalida = $row['fecha_check_out'];
    }

    if($fechaSalida != ''){
        $resultado["estado"] = "checkout";
        echo json_encode($resultado,JSON_UNESCAPED_UNICODE);

    } else {
        $resultado["estado"] = "checkin";
        echo json_encode($resultado,JSON_UNESCAPED_UNICODE);
    }
} else {
    $resultado["estado"] = "inicio";
    echo json_encode($resultado,JSON_UNESCAPED_UNICODE);
}

?>

```

- **registro / registro_checkin.php** → Realiza el check in.

```

registro > registro_checkin.php > ...
registro > registro_checkin.php > ...
1  ?>php
2  // Conexion a la base de datos
3  require '../conexion.php';
4  require '../funciones/funciones.php';
5
6  // Establecemos la zona horaria
7  date_default_timezone_set('Europe/Madrid');
8
9  // Obtenemos la fecha de hoy
10 $fechaLocal = strtotime(date('Y-m-d'));
11
12 // Valores obtenidos por post
13 $fechaEntrada = date('Y-m-d H:i:s',strtotime($_POST['fecha_check_in']));
14 $id_empleado = intval($_POST['id_empleado']);
15
16 // Consulta para comprobar si el check in está hecho
17 $consulta = $bd-&gt;query("SELECT * FROM presencialidades WHERE id_empleado=$id_empleado AND fecha='$fechaLocal'");
18
19 if (mysqli_num_rows($consulta) == 0) {
20     // Realizamos el check in
21     $consulta = $bd-&gt;query("INSERT INTO presencialidades (id_empleado,fecha,fecha_check_in) VALUES ($id_empleado,$fechaLocal)");
22
23     if(!$consulta){
24         printf("Error: %s\n", $bd-&gt;error);
25         exit();
26     } else {
27         $resultado["estado"] = "checkin";
28         echo json_encode($resultado,JSON_UNESCAPED_UNICODE);
29     }
30 }
31
32 ?&gt;
</pre

```

- **registro / registro_checkout.php** → Realiza el check out.

```
registro > registro_checkout.php > ...
1  <?php
2  // Conexion a la base de datos
3  require '../conexion.php';
4
5  // Establecemos la zona horaria
6  date_default_timezone_set('Europe/Madrid');
7
8  // Obtenemos la fecha de hoy
9  $fechaLocal = strval(date('Y-m-d'));
10
11 // Valores obtenidos por post
12 $fechaSalida = date('Y-m-d H:i:s', strtotime($_POST['fecha_check_out']));
13 $id_empleado = intval($_POST['id_empleado']);
14
15 // Consulta para comprobar si el check in est&aacute;a hecho para poder hacer el check out
16 $consulta = $bd->query("SELECT * FROM presencialidades WHERE id_empleado=$id_empleado AND fecha='$fechaLocal'");
17
18 if (mysqli_num_rows($consulta) > 0) {
19     // Realizamos el check out
20     $consulta = $bd->query("UPDATE presencialidades SET fecha_check_out = '$fechaSalida' WHERE id_empleado='$id_empleado'");
21
22     if(!$consulta){
23         printf("Error: %s\n", $bd->error);
24         exit();
25     } else {
26         $resultado["estado"] = "checkout";
27         echo json_encode($resultado,JSON_UNESCAPED_UNICODE);
28     }
29 }
30
31
32
33
```

- **configuracion / datos_usuario.php** → Cambio de contraseña.

```
datos_usuario.php > ...
configuracion > datos_usuario.php > ...
1  <?php
2  // Conexion a la base de datos
3  require '../conexion.php';
4
5  // Valores obtenidos por post
6  $usu_id = $_POST['id_empleado'];
7  $usu_nombre = $_POST['nombre'];
8  $usu_email = $_POST['email'];
9  $usu_password = $_POST['password'];
10 $usu_password_new = $_POST['new_password'];
11 $usu_password_verify = $_POST['verify_password'];
12
13 // Consulta si existe el usuario
14 $consulta = $bd->query("SELECT * FROM users WHERE email='$usu_email' AND id='$usu_id'");
15
16 if(!$consulta){
17     printf("Error: %s\n", $bd->error);
18     exit();
19 } else {
20     if($row = $consulta->fetch_assoc()){
21         // Verificamos la contrase&aacute;a actual
22         if(password_verify($usu_password,$row['password'])){
23             //Comprobamos que la nueva contrase&aacute;a sea igual a la verificada
24             if($usu_password_new == $usu_password_verify){
25                 //Encriptamos la contrase&aacute;a
26                 $passwordHash = password_hash($usu_password_new, PASSWORD_DEFAULT);
27                 //Actualizamos la contrase&aacute;a ya encriptada
28                 $consulta = $bd->query("UPDATE users SET `password` = '$passwordHash' WHERE email='$usu_email' AND id='$usu_id'");
29
30                 if(!$consulta){
31                     printf("Error: %s\n", $bd->error);
32                     exit();
33                 }
34             }
35         }
36     }
37 }
```

```

datos_usuario.php ×
configuracion > datos_usuario.php > ...
23     //Comprobamos que la nueva contraseña sea igual a la verificada
24     if($usu_password_new == $usu_password_verify){
25         //Encriptamos la contraseña
26         $passwordHash = password_hash($usu_password_new, PASSWORD_DEFAULT);
27         //Actualizamos la contraseña ya encriptada
28         $consulta = $bd->query("UPDATE users SET `password` = '$passwordHash' WHERE email='$usu_email' AND id='$usu_id'");
29
30         if(!$consulta){
31             printf("Error: %s\n", $bd->error);
32             exit();
33         } else {
34             echo "Contraseña actualizada con éxito";
35         }
36
37     } else {
38         echo "La nueva contraseña debe ser igual a la verificada";
39     }
40
41 } else {
42     echo "La contraseña actual es incorrecta";
43 }
44
45 }
46
47 ?:

```

INFORMACIÓN:

Los códigos de Productos, Consumos y Stocks son iguales al código de Voladuras.

Se insertan simultáneamente Productos y Stock. De la misma forma Consumos y Stock.

```

add_producto.php ×
productos > add_producto.php > ...
29     $voladura_id = $row[10];
30 }
31 }
32
33 // Comprobamos los valores
34 if(comprobarNumero($arena_06) && comprobarNumero($grava_612) && comprobarNumero($grava_1220) &&
35     comprobarNumero($grava_2023) && comprobarNumero($rechazo) && comprobarNumero($zahorra) &&
36     comprobarNumero($escollera) && comprobarNumero($mamposteria)){[
37
38     // Insertamos un nuevo producto
39     $consulta = $bd->query("INSERT INTO productos (arena_06,grava_612,grava_1220,grava_2023,rechazo,zahorra,escollera,mamposteria) VALUES ($arena_06,$grava_612,$grava_1220,$grava_2023,$rechazo,$zahorra,$escollera,$mamposteria)");
40
41     if(!$consulta){
42         printf("Error: %s\n", $bd->error);
43         exit();
44     } else {
45         echo 'Producto insertado con éxito';
46
47         // Insertamos el stock del producto
48         $consultaS = $bd->query("INSERT INTO stock_productos (arena_06,grava_612,grava_1220,grava_2023,rechazo,zahorra,escollera,mamposteria) VALUES ($arena_06,$grava_612,$grava_1220,$grava_2023,$rechazo,$zahorra,$escollera,$mamposteria)");
49
50         if(!$consultaS){
51             printf("Error: %s\n", $bd->error);
52             exit();
53         }// Insertamos a la vez el stock del producto
54     }
55
56 } else {
57     echo 'Los datos introducidos deben de ser números';
58 }
59
60
61 ]:

```

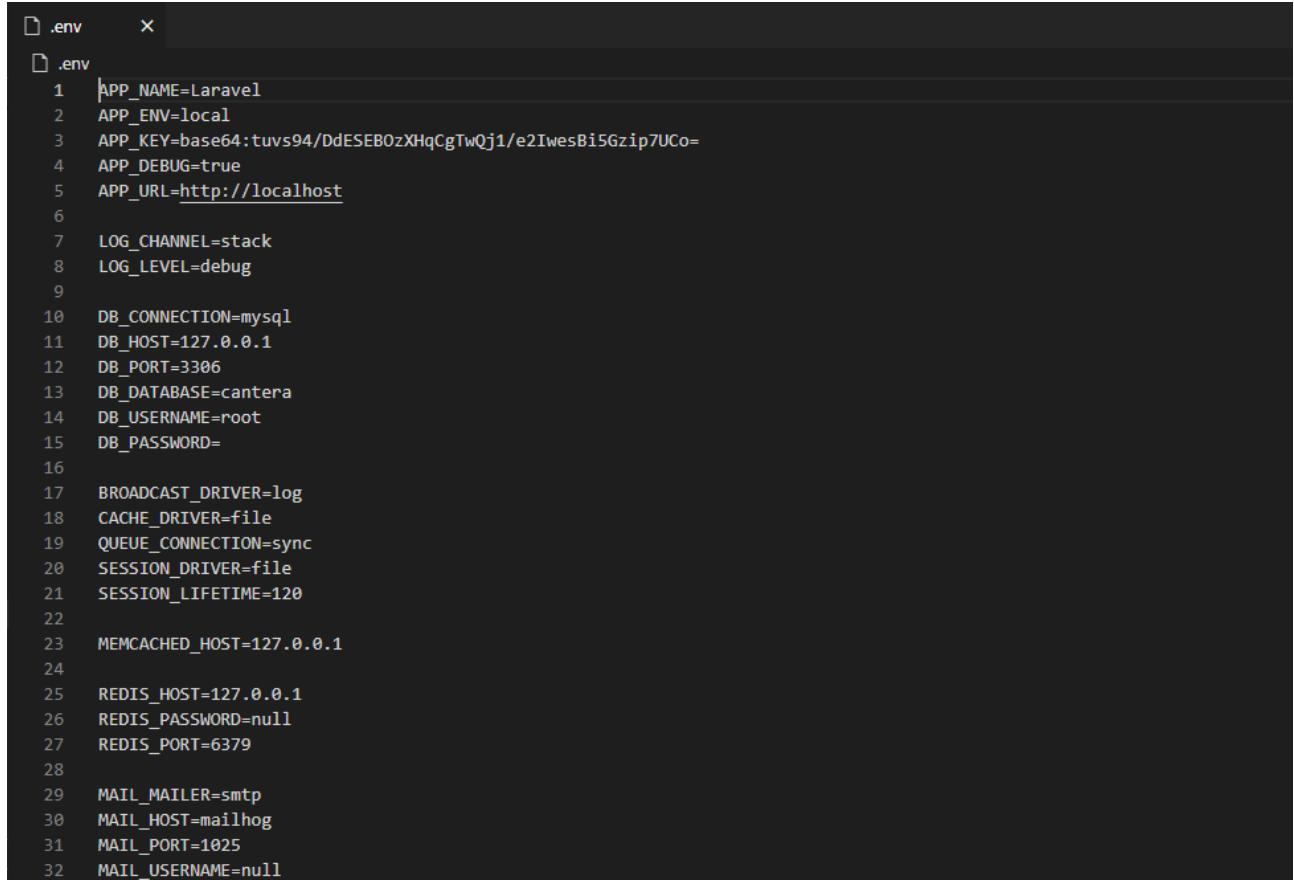
Los stock tienen un archivo mas, que es el que devuelve el último registro introducido. Nos sirve para que cuando vayamos a introducir uno de estos stock, se auto completen los campos con el último registro introducido, es más orientativo a la hora de actualizar el stock.

```
ultimo_add_stock_consumo.php X
consumos > stockconsumos > ultimo_add_stock_consumo.php > ...
1  ?>php
2  // Conexion a la base de datos
3  require '../conexion.php';
4
5  // Consulta que devuelve el ultimo stock de consumo ordenador por id
6  $consulta = $bd-&gt;query("SELECT * FROM stock_consumos ORDER BY id DESC LIMIT 1");
7
8  if(!$consulta){
9      printf("Error: %s\n", $bd-&gt;error);
10     exit();
11 } else {
12     if($row = $consulta-&gt;fetch_assoc()){
13
14         echo json_encode($row,JSON_UNESCAPED_UNICODE);
15     }
16 }
17
18
19
20 ? 
```

Código fuente Web

1.- ENV

En este archivo se configurará el modo en que se ejecuta nuestra aplicación, por defecto será el modo debug, además podemos configurar la conexión a la base de datos y la conexión con el servidor de correo electrónico.



```
□ .env      X
□ .env
1 APP_NAME=Laravel
2 APP_ENV=local
3 APP_KEY=base64:tuv94/DdESEB0zXHqCgTwQj1/e2IwesBi5Gzip7UCo=
4 APP_DEBUG=true
5 APP_URL=http://localhost
6
7 LOG_CHANNEL=stack
8 LOG_LEVEL=debug
9
10 DB_CONNECTION=mysql
11 DB_HOST=127.0.0.1
12 DB_PORT=3306
13 DB_DATABASE=cantera
14 DB_USERNAME=root
15 DB_PASSWORD=
16
17 BROADCAST_DRIVER=log
18 CACHE_DRIVER=file
19 QUEUE_CONNECTION=sync
20 SESSION_DRIVER=file
21 SESSION_LIFETIME=120
22
23 MEMCACHED_HOST=127.0.0.1
24
25 REDIS_HOST=127.0.0.1
26 REDIS_PASSWORD=null
27 REDIS_PORT=6379
28
29 MAIL_MAILER=smtp
30 MAIL_HOST=mailhog
31 MAIL_PORT=1025
32 MAIL_USERNAME=null
```

2.- database / migrations (Migraciones)

Las migraciones en Laravel son una herramienta que nos permite crear una especie de sistema de control de versiones de bases de datos donde podemos definir tablas con POO en vez de SQL, es compatible con los diferentes motores de base de datos dado que Laravel genera el SQL por nosotros, adaptado al tipo de base de datos con la que estemos trabajando. Laravel soporta: MySQL, Postgres, SQLite, y SQL Server.

```
2021_04_18_054157_create_voladuras_table.php ×  
database > migrations > 2021_04_18_054157_create_voladuras_table.php > ...  
1 <?php  
2  
3 use Illuminate\Database\Migrations\Migration;  
4 use Illuminate\Database\Schema\Blueprint;  
5 use Illuminate\Support\Facades\Schema;  
6  
7 class CreateVoladurasTable extends Migration  
8 {  
9     /**  
10      * Run the migrations.  
11      *  
12      * @return void  
13      */  
14     public function up()  
15     {  
16         Schema::create('voladuras', function (Blueprint $table) {  
17             $table->id();  
18             $table->string('localizacion');  
19             $table->double('m2_superficie');  
20             $table->string('malla_perforacion');  
21             $table->double('profundidad_barrenos');  
22             $table->integer('numero_barrenos');  
23             $table->double('kg_explotivo');  
24             $table->double('precio');  
25             $table->double('piedra_bruta');  
26             $table->dateTime('fecha_voladura');  
27             $table->integer('id_empleado');  
28             $table->timestamps();  
29         });  
30     }  
31  
32     /**
```

```
}
```

```
    /**  
     * Reverse the migrations.  
     *  
     * @return void  
     */  
    public function down()  
    {  
        Schema::dropIfExists('voladuras');  
    }
```

Explicación código migraciones: (Explicando una, se explican todas)

Ahora bien se puede observar que el archivo como tal no se llama simplemente **create_table_voladuras** sin **2021_04_18_054157_create_table_voladuras**, esto pasa porque Laravel al crear una migración agrega como prefijo la fecha y hora en la que fue creada la migración para poder ordenar qué migración va antes que otra. Además las migraciones que vienen por defecto en Laravel también se encuentran con este formato por lo cual podemos observar que estos dos archivos si tienen el mismo nombre.

Dentro de la estructura del archivo podemos ver dos funciones, una llamada **up()** y otra llamada **down()**, la primera función es en donde vamos a especificar la estructura de nuestra tabla, inicialmente y gracias al comando (**php artisan make:migration nombre_migracion**) se encuentran ya algunas cosas escritas como lo son la clase **Schema** en la cual se llama al método **create**, el cual nos permite crear la tabla en nuestra base de datos, esta recibe dos parámetros, el primero es el nombre que va a recibir la tabla y el segundo parámetro es una función closure o función anónima que lo que hace es definir las columnas de nuestra tabla, a su vez esta función anónima recibe como parámetro un objeto de tipo **Blueprint** que se agregó dentro del namespace con la palabra **use** en la cabecera del archivo, el objeto \$table es con el que vamos a trabajar para definir los campos, como se ve en la imagen anterior esto se logra escribiendo `$table->tipo_dato('nombre')`, esto puede variar dependiendo el tipo de dato que se use.

Y un campo de tipo timestamps sin nombre, el efecto que tendrá será agregar dos columnas muy útiles que son `created_at` y `updated_at` que son campos que se usan para (como su nombre lo dice) guardar el registro de cuando fue creado y cuando fue actualizado el registro.

La función **up** crea nuestra tabla en la base de datos, la función **down** lógicamente hace lo opuesto, y eso es eliminar la tabla de la base de datos, por eso dentro de esta función podemos observar que de la misma clase Schema se llama al método `drop` que significa dejar caer o dar de baja.

3.- routes / web.php:

Todas las rutas de Laravel se definen en los archivos que se encuentran en la carpeta `routes`. El framework carga estos archivos de forma automática. El archivo `routes/web.php` define las rutas para la interfaz web. A estas rutas se les asigna el grupo de `middleware=web`, el cual proporciona algunas características como el estado de la sesión y la protección CSRF.

```
web.php  X
routes > web.php > ...
1  <?php
2
3  use App\Http\Controllers\EmpleadoController;
4  use App\Http\Controllers\VoladuraController;
5  use App\Http\Controllers\ProductoController;
6  use App\Http\Controllers\StockProductoController;
7  use App\Http\Controllers\ConsumoController;
8  use App\Http\Controllers\StockConsumoController;
9  use App\Http\Controllers\PresencialidadController;
10 use Illuminate\Support\Facades\Route;
11 use App\Http\Controllers\HomeController;
12 use Illuminate\Support\Facades\Auth;
13
14 /*
15 |-----
16 | Web Routes
17 |-----
18 |
19 | Here is where you can register web routes for your application. These
20 | routes are loaded by the RouteServiceProvider within a group which
21 | contains the "web" middleware group. Now create something great!
22 |
23 */
24
25 Route::get('/', function () {
26     return view('welcome');
27     /* return view('index'); */
28 });
29
30 Auth::routes();
31
32 Route::get('/home', [HomeController::class, 'index'])->name('home');
```

```
web.php  X
routes > web.php > ...
33
34 Route::get('empleados', [EmpleadoController::class, 'index'])->name('empleados.index');
35
36 Route::get('empleados/create', [EmpleadoController::class, 'create'])->name('empleados.create');
37
38 Route::post('empleados', [EmpleadoController::class, 'store'])->name('empleados.store');
39
40 Route::get('empleados/{empleado}', [EmpleadoController::class, 'show'])->name('empleados.show');
41
42 Route::get('empleados/{empleado}/edit', [EmpleadoController::class, 'edit'])->name('empleados.edit');
43
44 // Se recomienda put para actualizar en vez de post
45 Route::put('empleados/{empleado}', [EmpleadoController::class, 'update'])->name('empleados.update');
46
47 // Ruta para eliminar un registro
48 Route::delete('empleados/{empleado}', [EmpleadoController::class, 'destroy'])->name('empleados.destroy');
49
50
51 Route::resource('voladuras', VoladuraController::class);
52
53 Route::resource('productos', ProductoController::class);
54
55 Route::resource('stock_productos', StockProductoController::class);
56
57 Route::resource('consumos', ConsumoController::class);
58
59 Route::resource('stock_consumos', StockConsumoController::class);
60
61 Route::resource('presencialidad', PresencialidadController::class);
62
63
```

Explicación código **web.php**

Tipos de rutas por encabezado Http

Las rutas están siempre declaradas usando la clase Route. Eso es lo que tenemos al principio, antes de :: . La parte get es el método que usamos para ‘capturar’ las peticiones que son realizadas usando el verbo ‘GET’ de HTTP hacia una URL concreta.

Como verás, todas las peticiones realizadas por un navegador web contienen un verbo. La mayoría de las veces, el verbo será GET , que es usado para solicitar una página web. Se envía una petición GET cada vez que escribes una nueva dirección web en tu navegador.

Aunque no es la única petición. También está POST , que es usada para hacer una petición y ofrecer algunos datos. Normalmente se usa para enviar un formulario en la que se necesita enviar los datos sin mostrarlo en la URL.

Hay otros verbos HTTP disponibles. He aquí algunos de los métodos que la clase de enrutado tiene disponible para ti:

- Route::get();
- Route::post();
- Route::any();
- Route::delete();
- Route::put();

Cualquier método de la clase Route recibe siempre dos argumentos, el primero es la URI con la que queremos hacer coincidir la URL y el segundo es la función a realizar que en este caso es un Closure que no es otra cosa que una función anónima, es decir, que no tiene un nombre o pueden ser relacionadas con métodos de un controlador

Podemos ver que hacemos uso de Route::resource, esta declaración de ruta única crea múltiples rutas para manejar una variedad de acciones en el recurso. Nos ahorra mucho código ya que de la otra manera para añadir, eliminar, etc.. teníamos que especificar put, delete... pero con resource todo esto te lo engloba.

4.- Modelo:

En Laravel podemos hacer uso de un **ORM** llamado Eloquent, un **ORM** es un **Mapeo Objeto-Relacional** por sus siglas en inglés (Object-Relational mapping), que es una forma de mapear los datos que se encuentran en la base de datos almacenados en un lenguaje de script SQL a objetos de PHP y viceversa, esto surge con la idea de tener un código portable con el que no tengamos la necesidad de usar lenguaje SQL dentro de nuestras clases de PHP. normalmente se almacenan en el directorio de aplicación **app/Models/**.

```
Voladura.php X
app > Models > Voladura.php > ...
1 <?php
2
3 namespace App\Models;
4
5 use Illuminate\Database\Eloquent\Factories\HasFactory;
6 use Illuminate\Database\Eloquent\Model;
7
8 class Voladura extends Model
9 {
10     use HasFactory;
11
12     protected $fillable = [
13         'localizacion',
14         'm2_superficie',
15         'malla_perforacion',
16         'profundidad_barrenos',
17         'numero_barrenos',
18         'kg_explotivo',
19         'precio',
20         'piedra_bruta',
21         'fecha_voladura',
22         'id_empleado'
23     ];
24 }
25
```

Explicación de código de los modelos.

Los modelos usan convenciones para que a Laravel se le facilite el trabajo y nos ahorre tanto líneas de código como tiempo para relacionar más modelos, las cuales son:

- El nombre de los modelos se escribe en singular, en contraste con las tablas de la BD que se escriben en plural.
- Usan notación UpperCamelCase para sus nombres.

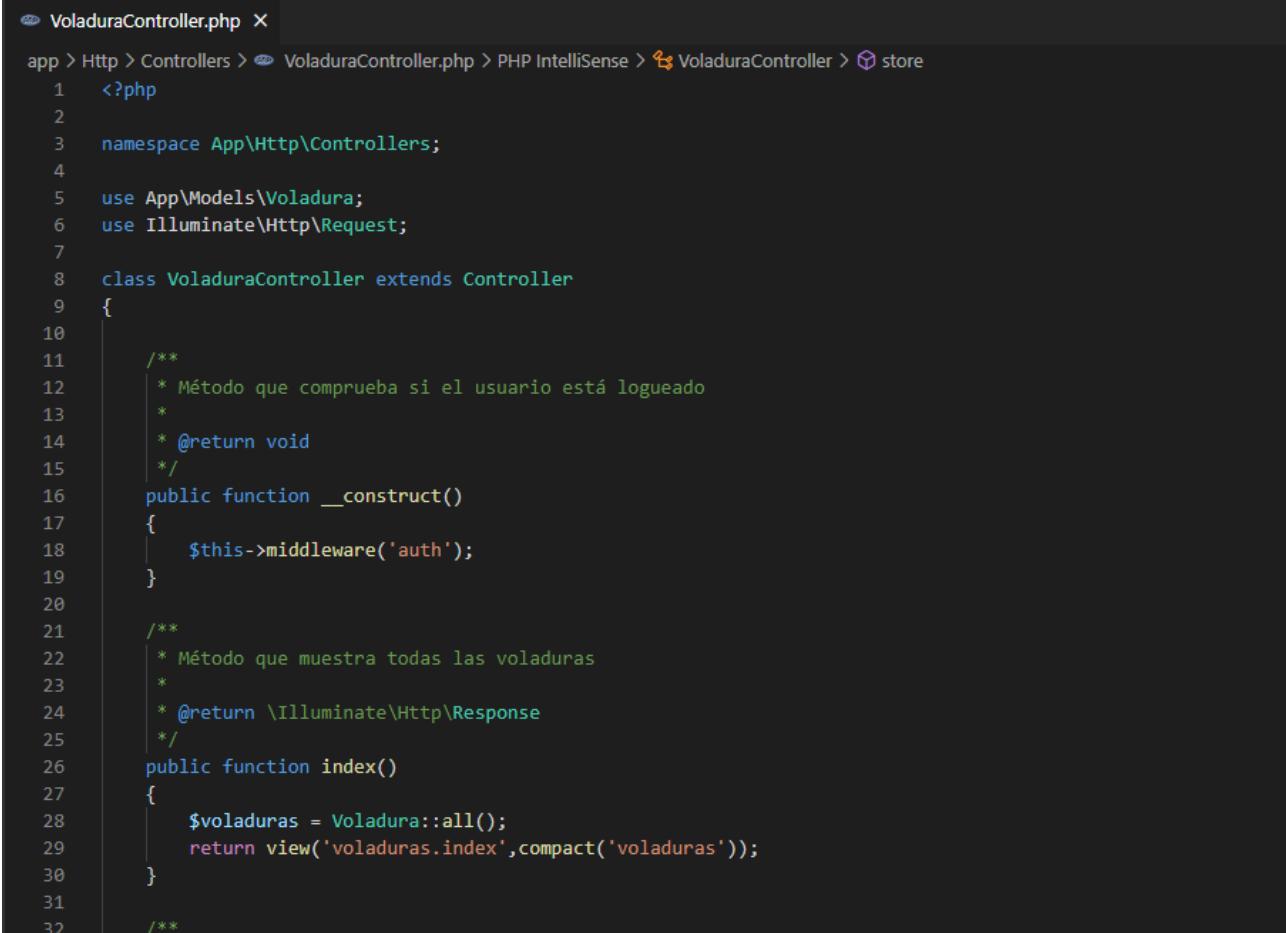
Estas convenciones nos ayudan a detectar automáticamente las tablas, por ejemplo: el modelo **Voladura** se encuentra en **singular** y con notación **UpperCamelCase** y para Laravel poder definir que tabla es la que esta ligada a este modelo le es suficiente con realizar la conversión a notación **underscore** y **plural**, dando como resultado la tabla: **voladuras**.

protected \$fillable = [] .

Laravel permite asignar valores a cada uno de los atributos de la tabla de una manera muy sencilla con un **array** asociativo de elementos (que pueden ser los campos de un formulario), de esta forma se facilita el proceso de almacenaje desde un **form**.

5.- Controlador

Los **Controladores** puede agrupar las peticiones HTTP relacionada con la manipulación lógica en una clase. Los **Controladores** normalmente se almacenan en el directorio de aplicación **app/Http/Controllers/**.



```
VoladuraController.php X
app > Http > Controllers > VoladuraController.php > PHP IntelliSense > VoladuraController > store
1  <?php
2
3  namespace App\Http\Controllers;
4
5  use App\Models\Voladura;
6  use Illuminate\Http\Request;
7
8  class VoladuraController extends Controller
9  {
10
11     /**
12      * Método que comprueba si el usuario está logueado
13      *
14      * @return void
15      */
16     public function __construct()
17     {
18         $this->middleware('auth');
19     }
20
21     /**
22      * Método que muestra todas las voladuras
23      *
24      * @return \Illuminate\Http\Response
25      */
26     public function index()
27     {
28         $voladuras = Voladura::all();
29         return view('voladuras.index',compact('voladuras'));
30     }
31
32     /**
33      * Método que almacena una nueva voladura
34      *
35      * @param Request $request
36      * @return \Illuminate\Http\Response
37      */
38     public function store(Request $request)
39     {
40         $voladura = new Voladura();
41         $voladura->fill($request->all());
42         $voladura->save();
43
44         return redirect('/voladuras');
45     }
46 }
```

```

app > Http > Controllers > VoladuraController.php > PHP Intellisense > VoladuraController > store
 32 /**
 33  * Método que nos devuelve la view create (form) para poder añadir una nueva voladura
 34  *
 35  * @return \Illuminate\Http\Response
 36  */
 37 public function create()
 38 {
 39     // Devolvemos la view create
 40     return view('voladuras.create');
 41 }
 42
 43 /**
 44  * Método que obtiene los valores insertados por el usuario, para ser añadidos a la base de datos
 45  *
 46  * @param \Illuminate\Http\Request $request
 47  * @return \Illuminate\Http\Response
 48  */
 49 public function store(Request $request)
 50 {
 51     //Validamos los campos
 52     $request->validate([
 53         'localizacion'=>['required'],
 54         'superficie'=>['required','numeric','between:0,99999.99'],
 55         'perforacion'=>['required'],
 56         'profundidadBarrenos'=>['required','numeric','between:0,99999.99'],
 57         'numeroBarrenos'=>['required','numeric'],
 58         'explosivo' => ['required','numeric','between:0,99999.99'],
 59         'precio'=>['required','numeric','between:0,99999.99'],
 60         'piedraBruta'=>['required','numeric','between:0,99999.99'],
 61         'fechaVoladura'=>['required']
 62     ]);
 63 }

```

```

app > Http > Controllers > VoladuraController.php > PHP Intellisense > VoladuraController > store
 65     // Creamos una nueva voladura
 66     Voladura::create([
 67         'localizacion'=>$request->localizacion,
 68         'm2_superficie'=>$request->superficie,
 69         'malla_perforacion'=>$request->perforacion,
 70         'profundidad_barrenos'=>$request->profundidadBarrenos,
 71         'numero_barrenos'=>$request->numeroBarrenos,
 72         'kg_explotivo' => $request->explosivo,
 73         'precio'=>$request->precio,
 74         'piedra_bruta'=>$request->piedraBruta,
 75         'fecha_voladura'=> date('Y-m-d H:i:s', strtotime($request->fechaVoladura)) /* date('Y-m-d', strtotime($request->fechaVoladura)) */,
 76         'id_empleado'=> auth()->user()->id,
 77     ]);
 78
 79     // Redirigimos al index donde nos mostrarán todos las voladuras inclusive el añadido
 80     return redirect()->route('voladuras.index')->with('Mensaje','Voladura creada con éxito');
 81 }
 82
 83 /**
 84  * Método que nos devuelve la ficha de información sobre la fila seleccionada
 85  *
 86  * @param Voladura $voladura
 87  * @return \Illuminate\Http\Response
 88  */
 89 public function show(Voladura $voladura)
 90 {
 91     //Al view show, pasamos como parametro la voladura seleccionada
 92     return view('voladuras.show',compact('voladura'));
 93 }
 94
 95 /**
 96 */

```

```
@@ VoladuraController.php X
app > Http > Controllers > @@ VoladuraController.php > PHP IntelliSense > VoladuraController > store
96     /**
97      * Método que nos devuelve la view edit (form) para poder actualizar una voladura
98      *
99      * @param Voladura $voladura
100     * @return \Illuminate\Http\Response
101    */
102   public function edit(Voladura $voladura)
103   {
104       //Al view edit, pasamos como parametro la voladura seleccionada
105       return view('voladuras.edit',compact('voladura'));
106   }
107
108 /**
109  * Método que obtiene los valores insertados por el usuario, para ser actualizados sobre la fila seleccionada
110 *
111 * @param \Illuminate\Http\Request $request
112 * @param Voladura $voladura
113 * @return \Illuminate\Http\Response
114 */
115 public function update(Request $request, Voladura $voladura)
116 {
117     //Validamos los campos
118     $request->validate([
119         'localizacion'=>'required',
120         'superficie'=>'required','numeric','between:0,99999.99'],
121         'perforacion'=>'required'],
122         'profundidadBarrenos'=>'required','numeric','between:0,99999.99'],
123         'numeroBarrenos'=>'required','numeric'],
124         'explosivo' => ['required','numeric','between:0,99999.99'],
125         'precio'=>['required','numeric','between:0,99999.99'],
126         'piedraBruta'=>['required','numeric','between:0,99999.99'],
127         'fechaVoladura'=>['required']
```

```
@@ VoladuraController.php X
app > Http > Controllers > @@ VoladuraController.php > PHP IntelliSense > VoladuraController > store
129
130     // Actualizamos la voladura con los datos obtenidos por request
131     $voladura->update([
132         'localizacion'=>$request->localizacion,
133         'm2_superficie'=>$request->superficie,
134         'malla_perforacion'=>$request->perforacion,
135         'profundidad_barrenos'=>$request->profundidadBarrenos,
136         'numero_barrenos'=>$request->numeroBarrenos,
137         'kg_explotivo' => $request->explosivo,
138         'precio'=>$request->precio,
139         'piedra_bruta'=>$request->piedraBruta,
140         'fecha_voladura'=> date('Y-m-d H:i:s', strtotime($request->fechaVoladura)),
141         'id_empleado'=> auth()->user()->id,
142     ]);
143
144     // Redirigimos a index
145     return redirect()->route('voladuras.index')->with('Mensaje', 'Voladura modificada con éxito');
146 }
147
148 /**
149  * Método que elimina la fila seleccionada
150 *
151 * @param Voladura $voladura
152 * @return \Illuminate\Http\Response
153 */
154 public function destroy(Voladura $voladura)
155 {
156     // Eliminamos
157     $voladura->delete();
158     //Redirigimos a index
159     return redirect()->route('voladuras.index');
160 }
161 }
```

Explicación del código del **Controlador**:

Un controller usualmente trabaja con las peticiones:

- **GET.**
- **POST.**
- **PUT.**
- **DELETE.**
- **PATCH.**

Asociando los métodos de la siguiente forma:

- **GET:** index, create, show, edit.
- **POST:** store.
- **PUT:** update.
- **DELETE:** destroy.
- **PATCH:** update.

Los controladores nos ayudan a agrupar estas peticiones en una clase que se liga a las rutas, en el archivo routes/web.php, para esto usamos un tipo de ruta resource. (Route::resource)

Esta ruta nos creara un grupo de rutas de recursos con las peticiones que estas mencionadas arriba: **index, create, show, edit, store, update, destroy**. Estas son las operaciones mas usadas en una clase y para no tener que crear una ruta para cada método Laravel agrupa todo esto con una ruta de tipo resource que se liga a un controlador.

Estos métodos significan:

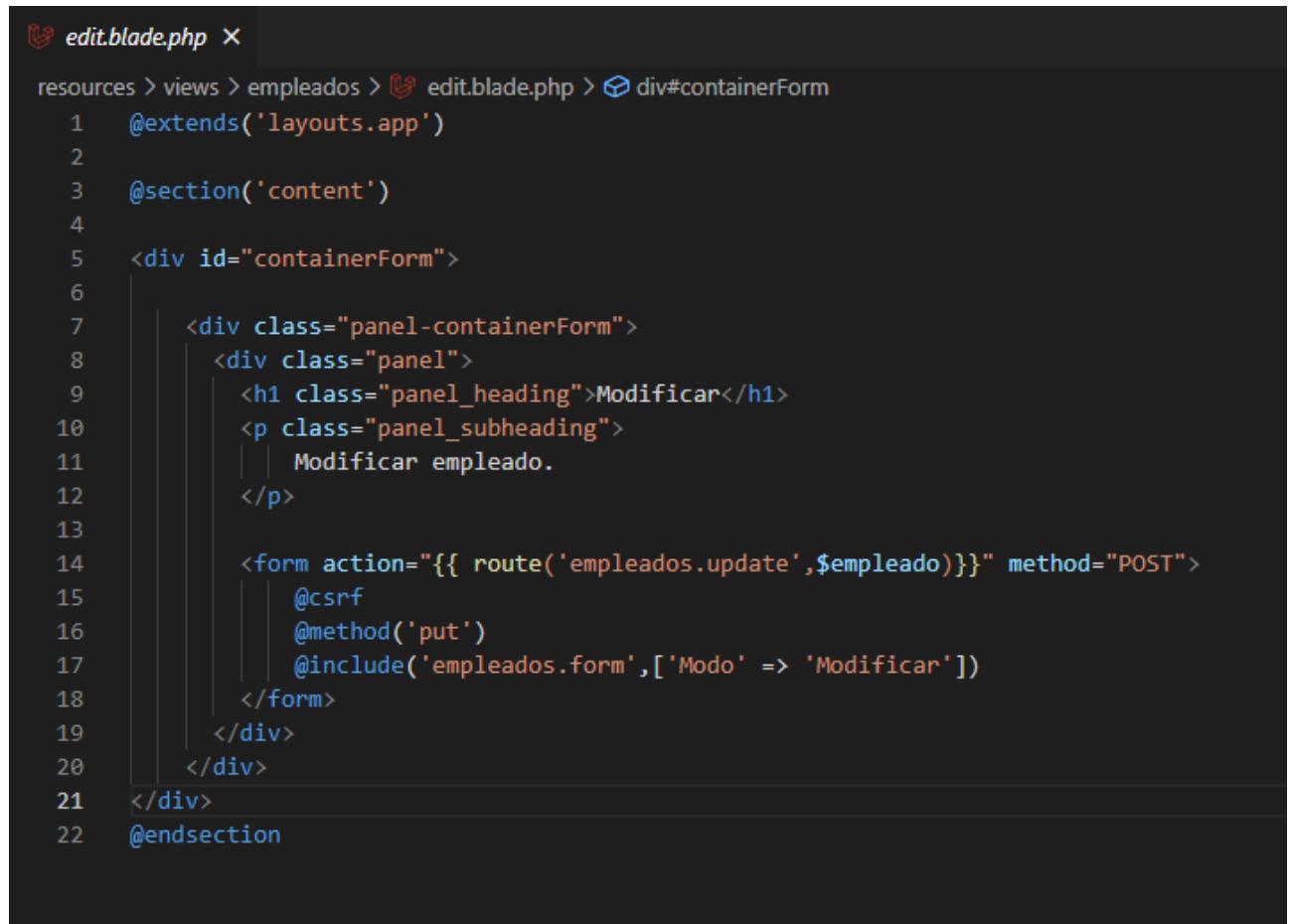
- **index:** Es el método inicial de la ruta resource. Lo usamos para mostrar una vista como página principal en la que mostramos todos los registros de una tabla o bien no es necesario mostrar información y solo tener la función de página de inicio.
- **create:** Este método lo usamos para direccionar el sistema a la vista donde se van a recolectar los datos(formulario) para después almacenarlos en un registro nuevo, y normalmente redirigirá al index.
- **show:** Hacemos una consulta sobre un elemento de la base de datos para realizar una descripción.
- **edit:** Este método es similar al de **create** porque lo podemos usar para mostrar una vista que recolecta los datos pero a diferencia de create es con el fin de actualizar un registro.
- **store:** Aquí es donde se actualiza un registro en específico que proviene del método create y normalmente redirige al index. Además hacemos la validaciones de los datos recibidos por request.
- **update:** Al igual que el store, solo que en vez de provenir de create proviene de **edit** y en vez de crear un nuevo registro, busca uno existente y lo modifica, también suele redirigir al index. Y como hemos dicho en store, también se validan los datos.

- **destroy:** En este método usualmente se destruye o elimina un registro y la petición puede provenir de donde sea siempre y cuando sea llamado con el método **DELETE**, después puede redirigir al index o a otro sitio dependiendo si logró eliminar o no.

6.- Blade

Blade es un simple pero poderoso motor de plantillas incluido con Laravel. A diferencia de otros populares motores de plantillas para PHP, Blade no limita el uso de código PHP simple en las vistas. Las vistas en Blade se compilan a código PHP y se cachean hasta que son modificadas, básicamente esto se traduce en que Blade añade sobrecarga cero a las aplicaciones. Las vistas en Blade usan la extensión .blade.php y normalmente se almacenan en el directorio resources/views.

Las capturas de ejemplo son edit.blade.php y form.blade.php (Muestro solo una pequeña parte del form)



```

edit.blade.php ×

resources > views > empleados > edit.blade.php > div#containerForm
1  @extends('layouts.app')
2
3  @section('content')
4
5  <div id="containerForm">
6
7      <div class="panel-containerForm">
8          <div class="panel">
9              <h1 class="panel_heading">Modificar</h1>
10             <p class="panel_subheading">
11                 Modificar empleado.
12             </p>
13
14             <form action="{{ route('empleados.update', $empleado) }}" method="POST">
15                 @csrf
16                 @method('put')
17                 @include('empleados.form', ['Modo' => 'Modificar'])
18             </form>
19         </div>
20     </div>
21 </div>
22 @endsection

```

```

form.blade.php ×

resources > views > empleados > form.blade.php > div.datos
1  @section('breadcrumb')
2      <li class="breadcrumb-item">
3          <a href="{{ route('empleados.index') }}">Empleados</a>
4      </li>
5      <li class="breadcrumb-item active" aria-current="page">
6          @if ($Modo == 'Registrar') Registrar Empleado @else Modificar Empleado @endif
7      </li>
8  @endsection
9
10
11 <div class="nameForm">
12     <div class="input">
13         <label for="nombre">Nombre</label>
14         <input type="text" id="nombre" name="nombre" value="{{ isset($empleado->nombre)? old('nombre', $empleado->nombre) : null }}"/>
15         <div class="input_indicator"></div>
16         @error('nombre')
17             <br>
18             <small class="mensaje">*{{ $message }}</small>
19         @enderror
20     </div>
21     <div class="input">
22         <label for="apellidos">Apellidos</label>
23         <input type="text" id="apellidos" name="apellidos" value="{{ isset($empleado->apellidos)? old('apellidos', $empleado->apellidos) : null }}"/>
24         <div class="input_indicator"></div>
25         @error('apellidos')
26             <br>
27             <small class="mensaje">*{{ $message }}</small>
28         @enderror
29     </div>
30
31     <div class="input">
32         <label for="dni">Dni</label>
33         <input type="text" id="dni" name="dni" value="{{ isset($empleado->dni)? old('dni', $empleado->dni) : null }}"/>

```

Explicación código **Blade**:

En los formularios de Laravel es necesario indicar la ruta, añadir @csrf y especificar como quieres que se envíen los datos.

Las falsificaciones de solicitudes entre sitios son un tipo de exploit malicioso mediante el cual se ejecutan comandos no autorizados en nombre de un usuario autenticado. Afortunadamente, Laravel facilita la protección de su aplicación contra ataques de falsificación de solicitudes entre sitios, por ello se implementa @csrf en los formularios.

En la segunda captura muestro como sería el formulario, @error se usa para capturar los mensajes de validación de los campos.

Blade es un motor de plantillas simples pero poderoso provisto con Laravel. A diferencia de los diseños de controladores , Blade se basa en la herencia y las secciones de la plantilla. Tolas las plantillas Blade deben usar la .blade. Php extensión. Para estas herencias hacemos uso de @section , @extends, @yield

INFORMACIÓN:

Para las capturas he mostrado solamente una clase, ya que todas tienen la misma estructura de código.

9.- Manual de configuración y funcionamiento de la aplicación.

- Manual de configuración de la App de Android

Requisitos :

- Poseer un teléfono móvil con Sistema Operativo de Android.
- Versión de Android superior o igual a la 4.0.
- Descargar App de la Play Store.
- (Próximamente para IOS)

- Manual de configuración de la Web

Requisitos :

- Software Windows o Ubuntu
- Descargar Xampp, este es una distribución de Apache completamente gratuita y fácil de instalar que contiene MariaDB, PHP y Perl.

Apache: Es capaz de alojar las páginas que creemos y proporcionar el acceso y funcionalidad a ellas por clientes externos o en una red local.

MySQL: es el sistema gestor libre de bases de datos SQL. Proporcionará la capacidad al servidor web de establecer un enlace de consulta a una base de datos local que almacenará datos de la página web y los servicios de hosting. El paquete dispone del **cliente SQL MariaDB**.

PHP: es el paquete encargado de “entender” las páginas web creadas. PHP es el lenguaje de programación más extensamente utilizado para la creación de páginas webs dinámicas.

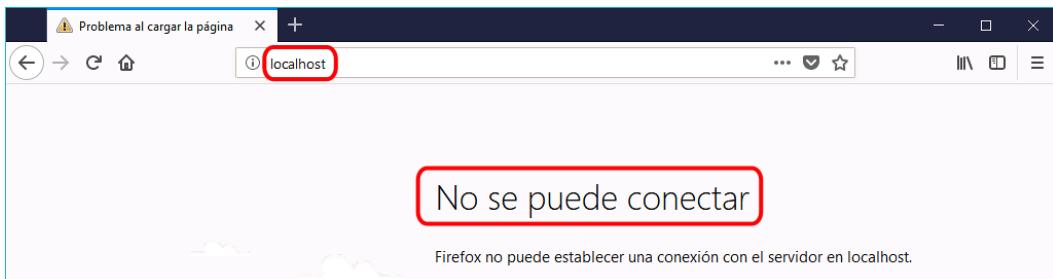
Descargar Xampp

Vamos a la página oficial de xamp y descargamos una versión 7.3.28.

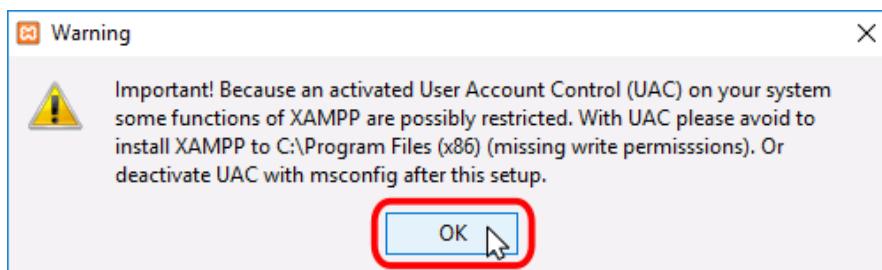
Versión	Suma de comprobación	Tamaño
7.3.28 / PHP 7.3.28 ¿Qué está incluido?	md5 sha1	Descargar (64 bit) 155 Mb

Instalación de Xampp

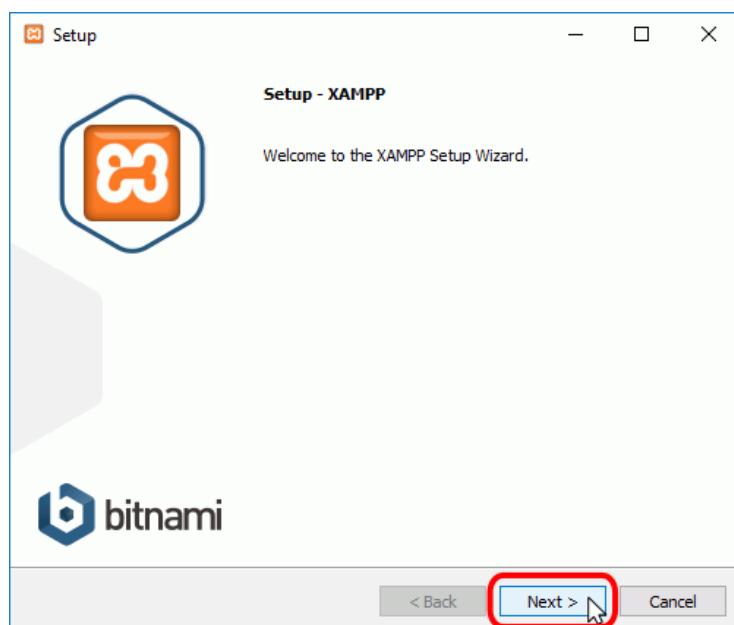
Antes de instalar un servidor de páginas web es conveniente comprobar si no hay ya uno instalado, o al menos si no está en funcionamiento. Para ello, es suficiente con abrir el navegador y escribir la dirección **http://localhost**. Si se obtiene un mensaje de error es que no hay ningún servidor de páginas web en funcionamiento (aunque podría haber algún servidor instalado, pero no estar en funcionamiento).



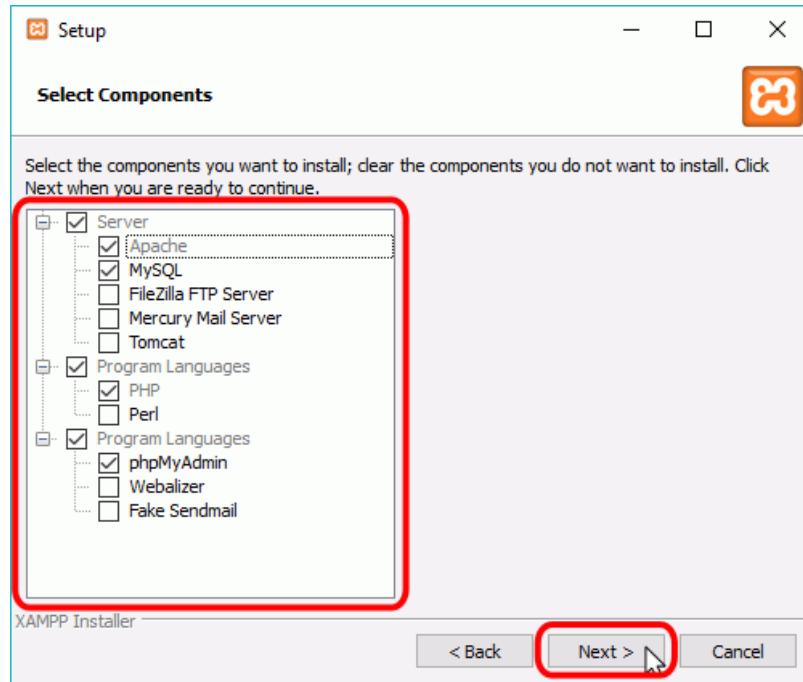
Una vez obtenido el archivo de instalación de XAMPP, hay que hacer doble clic sobre él para ponerlo en marcha. Al poner en marcha el instalador XAMPP nos muestra un aviso que aparece si está activado el Control de Cuentas de Usuario y recuerda que algunos directorios tienen permisos restringidos:



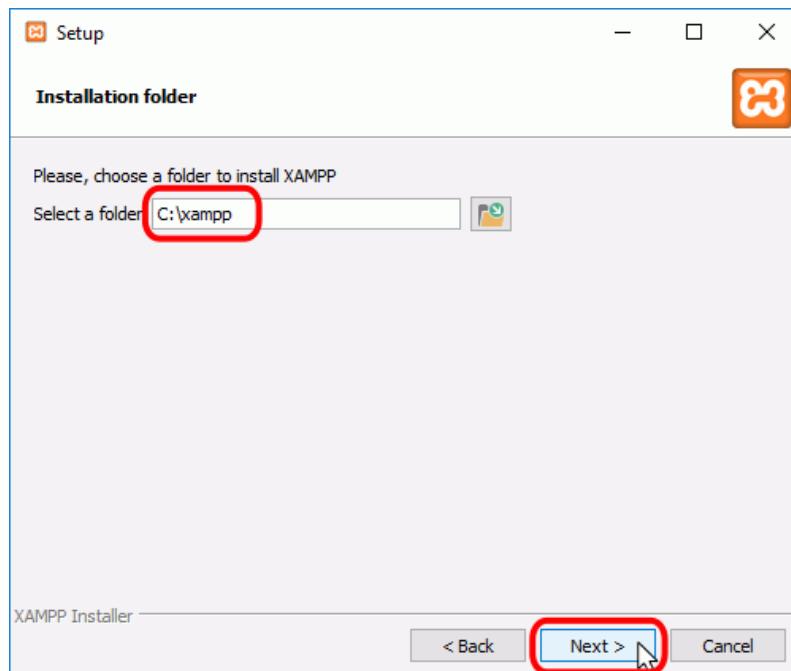
A continuación se inicia el asistente de instalación. Para continuar, haga clic en el botón "Next".



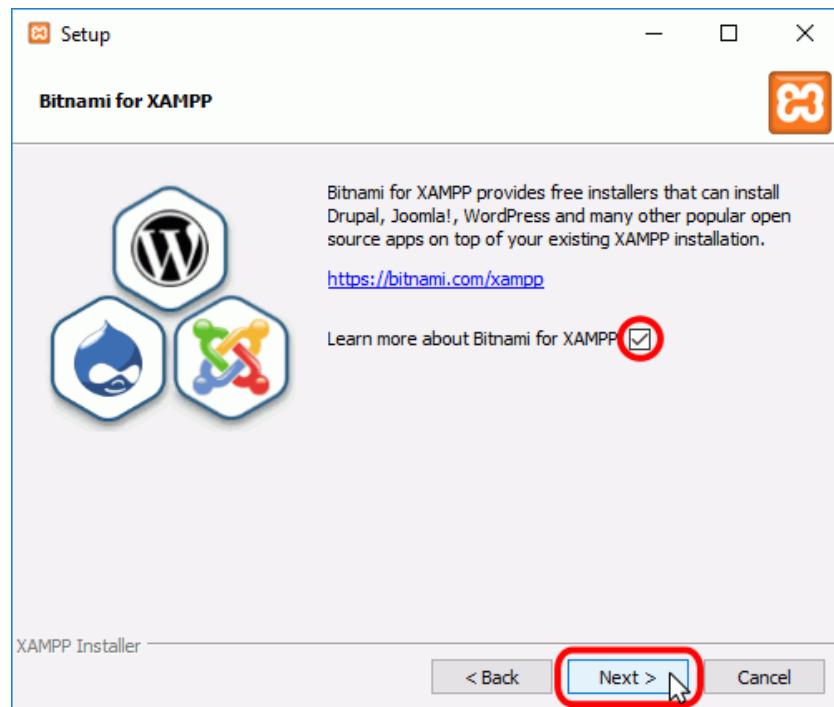
Los componentes mínimos que instala XAMPP son el servidor Apache y el lenguaje PHP, pero XAMPP también instala otros elementos. En la pantalla de selección de componentes puede elegir la instalación o no de estos componentes. Para seguir estos apuntes se necesita al menos instalar MySQL y phpMyAdmin.



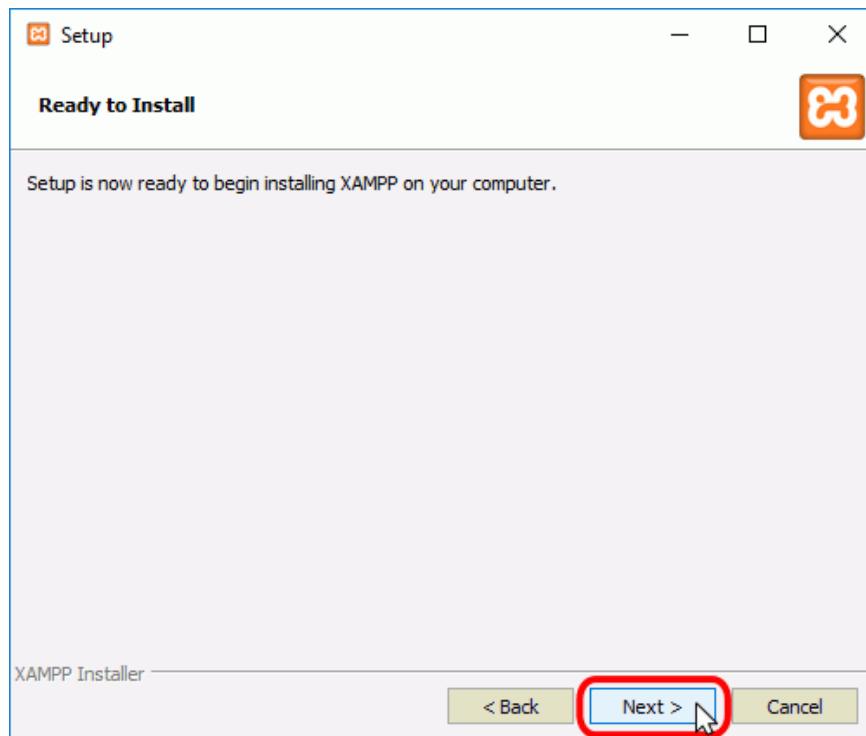
En la siguiente pantalla puede elegir la carpeta de instalación de XAMPP. La carpeta de instalación predeterminada es C:\xampp. Si quiere cambiarla, haga clic en el icono de carpeta y seleccione la carpeta donde quiere instalar XAMPP. Para continuar la configuración de la instalación, haga clic en el botón "Next".



La siguiente pantalla ofrece información sobre los instaladores de aplicaciones para XAMPP creados por Bitnami. Haga clic en el botón "Next" para continuar. Si deja marcada la casilla, se abrirá una página web de Bitnami en el navegador.



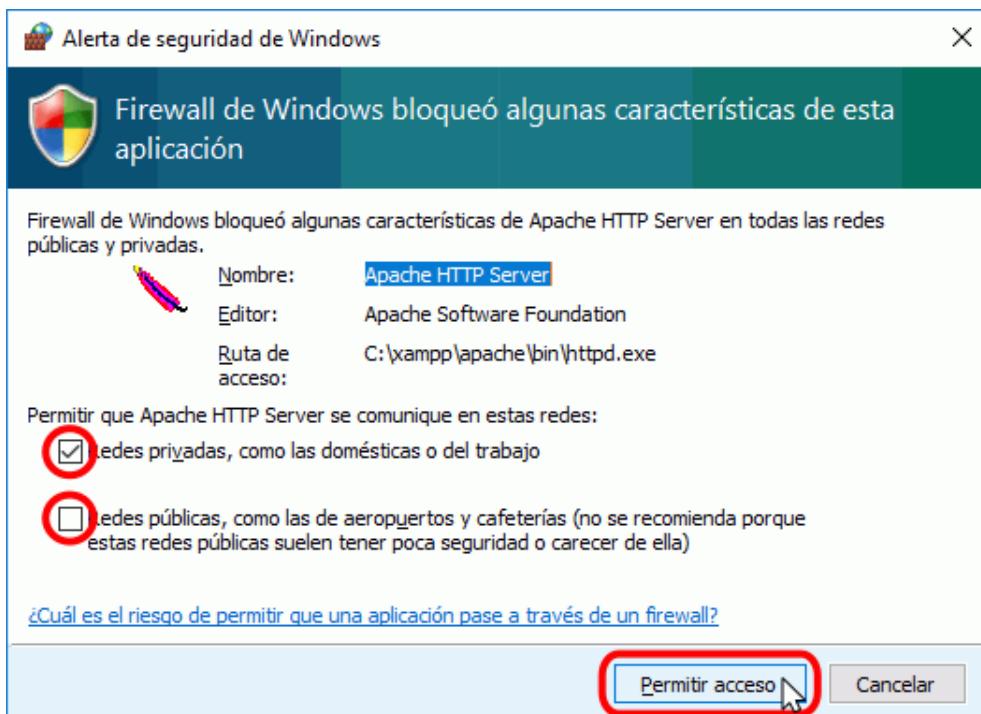
Una vez elegidas las opciones de instalación en las pantallas anteriores, esta pantalla es la pantalla de confirmación de la instalación. Haga clic en el botón "Next" para comenzar la instalación en el disco duro.



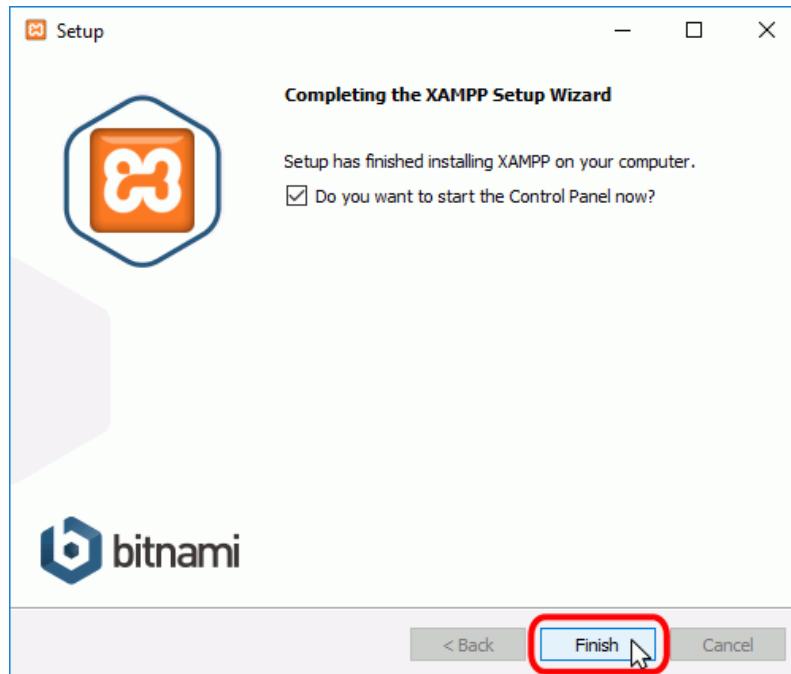
El proceso de copia de archivos puede durar unos minutos.



Durante la instalación, si en el ordenador no se había instalado Apache anteriormente, en algún momento se mostrará un aviso del cortafuegos de Windows para autorizar a Apache a comunicarse en las redes privadas o públicas. Una vez elegidas las opciones deseadas (en estos apuntes se recomienda permitir las redes privadas y denegar las redes públicas), haga clic en el botón "Permitir acceso".



Una vez terminada la copia de archivos, la pantalla final confirma que XAMPP ha sido instalado. Si se deja marcada la casilla, se abrirá el panel de control de XAMPP. Para cerrar el programa de instalación, haga clic en el botón "Finish".

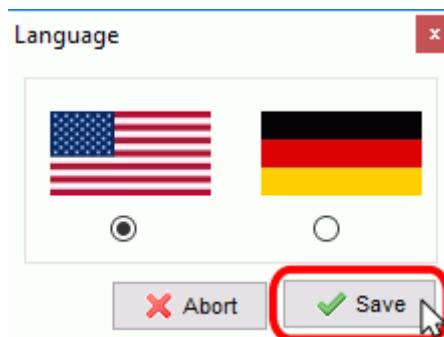


El Panel de Control de XAMPP

Abrir y cerrar el panel de control

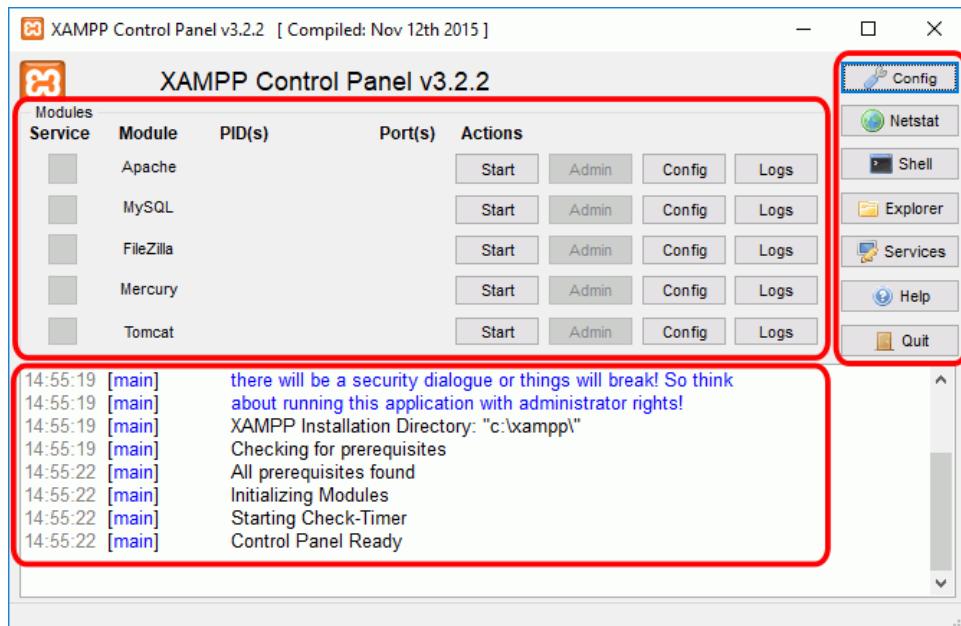
Al panel de control de XAMPP se puede acceder mediante el menú de inicio "Todos los programas > XAMPP > XAMPP Control Panel" o, si ya está iniciado, mediante el ícono del área de notificación.

La primera vez que se abre el panel de control de XAMPP, se muestra una ventana de selección de idioma que permite elegir entre inglés y alemán.

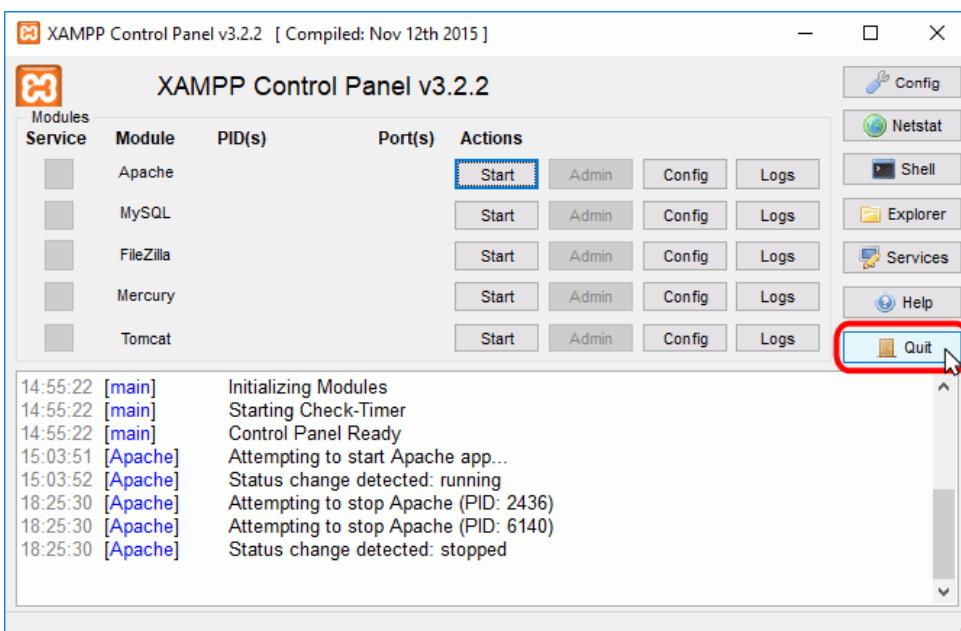


El panel de control de XAMPP se divide en tres zonas:

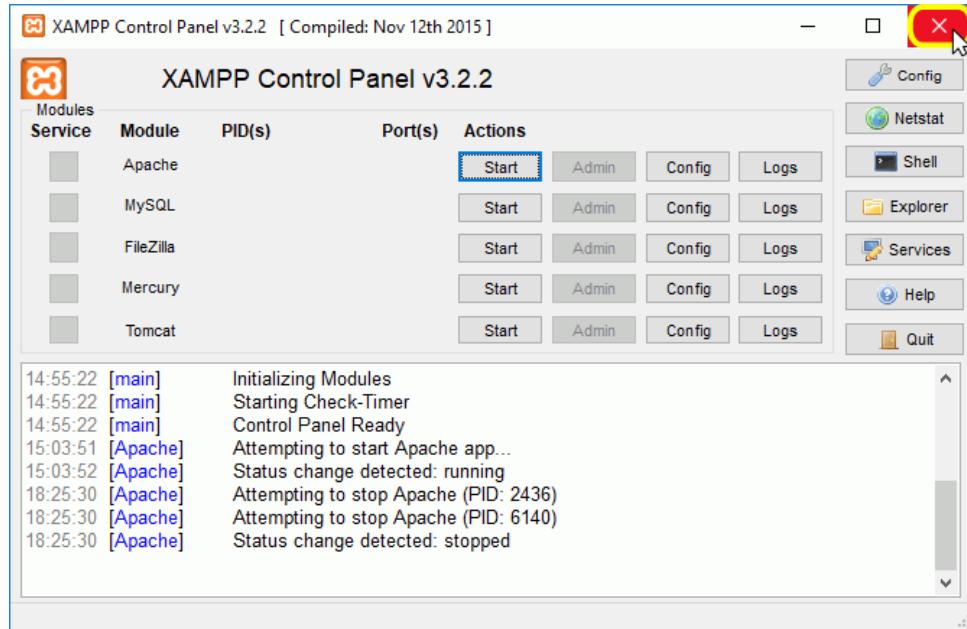
- La zona de módulos, que indica para cada uno de los módulos de XAMPP: si está instalado como servicio, su nombre, el identificador de proceso, el puerto utilizado e incluye unos botones para iniciar y detener los procesos, administrarlos, editar los archivos de configuración y abrir los archivos de registro de actividad.
- La zona de notificación, en la que XAMPP informa del éxito o fracaso de las acciones realizadas.
- La zona de utilidades, para acceder rápidamente.



Para cerrar el panel de control de XAMPP hay que hacer clic en el botón **Quit** (al cerrar el panel de control no se detienen los servidores):



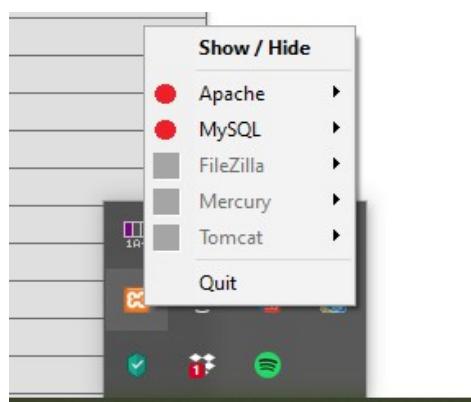
El botón Cerrar en forma de aspa no cierra realmente el panel de control, sólo lo minimiza:



Si se ha minimizado el panel de control de XAMPP, se puede volver a mostrar haciendo doble clic en el ícono de XAMPP del área de notificación.

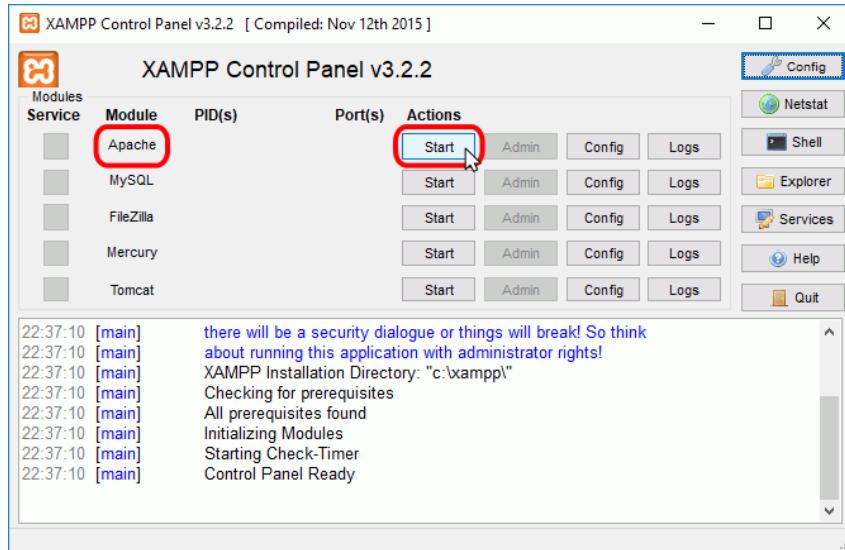


Haciendo clic derecho en el ícono de XAMPP del área de notificación se muestra un menú que permite mostrar u ocultar el panel de control, arrancar o detener servidores o cerrar el panel de control.

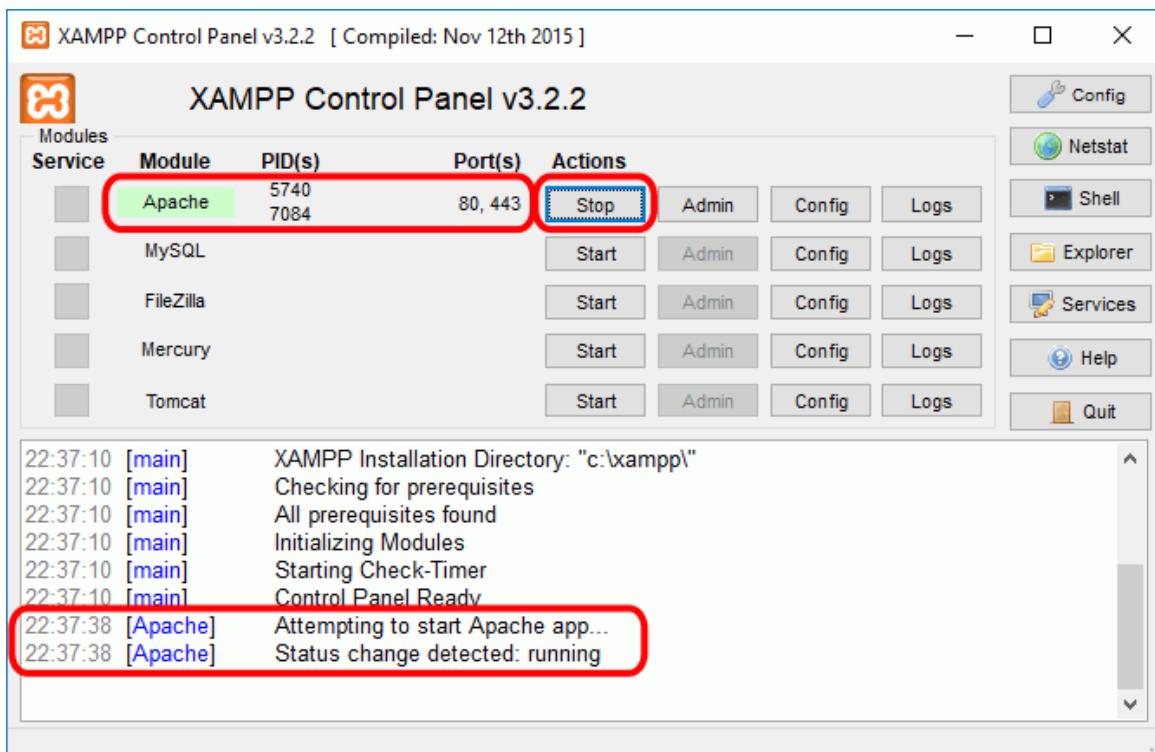


Iniciar servidores

Para poner en funcionamiento Apache (u otro servidor), hay que hacer clic en el botón "Start" correspondiente:

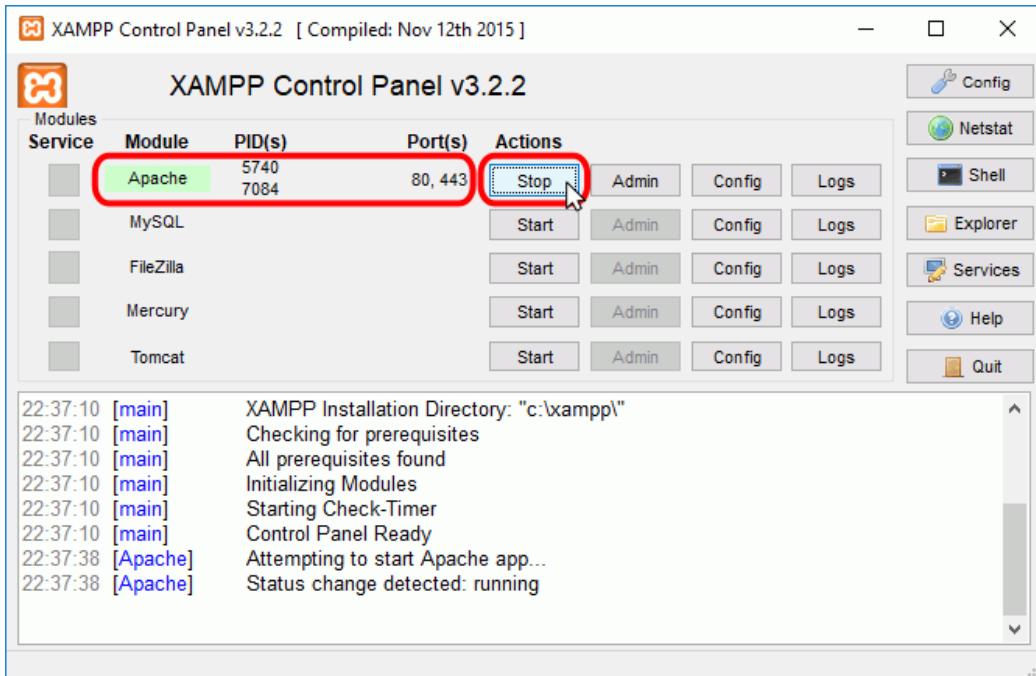


Si el arranque de Apache tiene éxito, el panel de control mostrará el nombre del módulo con fondo verde, su identificador de proceso, los puertos abiertos (http y https), el botón "Start" se convertirá en un botón "Stop" y en la zona de notificación se verá el resultado de las operaciones realizadas.

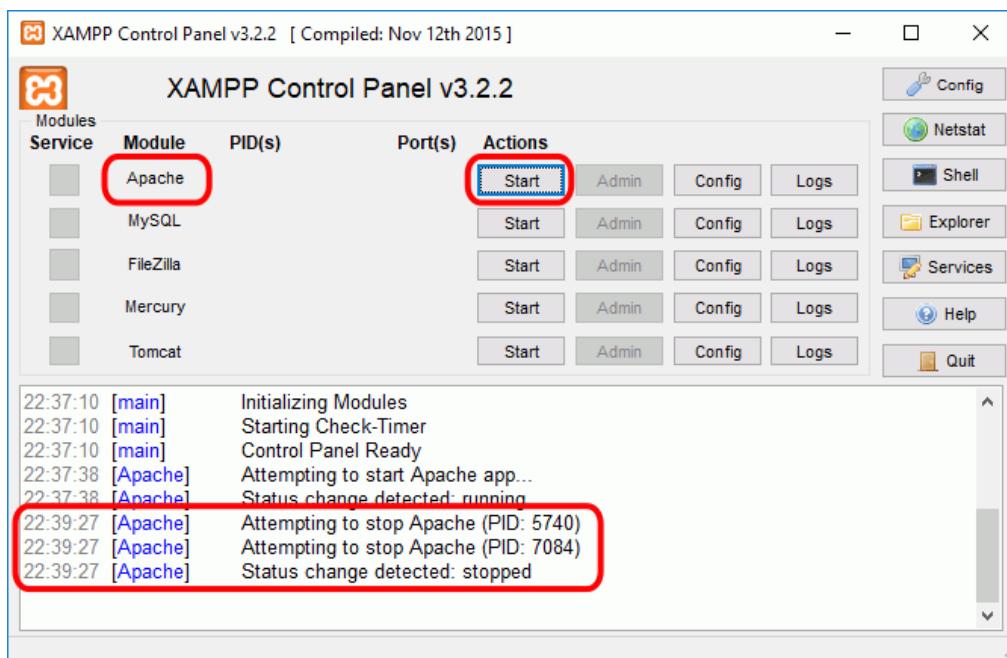


Detener servidores

Para detener Apache (u otro servidor), hay que hacer clic en el botón "Stop" correspondiente a Apache.



Si la parada de Apache tiene éxito, el panel de control mostrará el nombre del módulo con fondo gris, sin identificador de proceso ni puertos abiertos (http y https), el botón "Stop" se convertirá en un botón "Start" y en la zona de notificación se verá el resultado de las operaciones realizadas.



La instalación de xampp ha terminado. Para comprobar si ha sido correcta vuelva a escribir en la url del navegador localhost, aparecerá el panel administrativo de xampp.

Base de datos

La base de datos tiene que ser importada dentro de phpMyAdmin, para ello pondremos en la url localhost, aparecerá la pantalla administrativa de xampp, en la esquina superior derecha se encuentra la opción phpMyAdmin, pinchamos y se nos abrirá una nueva ventana.

Esta ventana es una interfaz diseñada para manejar la administración y gestión de bases de datos MySQL.

Para importar la base de datos, pinchamos en la opción de menú importar.



Buscamos el archivo y damos a continuar.

Importando al servidor actual

Archivo a importar:

El archivo puede ser comprimido (gzip, bzip2, zip) o descomprimido.

A compressed file's name must end in .[format].[compression]. Example: .sql.zip

Buscar en su ordenador: No se ha seleccionado ningún archivo (Máximo: 40MB)

También puede arrastrar un archivo en cualquier página.

Conjunto de caracteres del archivo:

La base de datos está instalada.

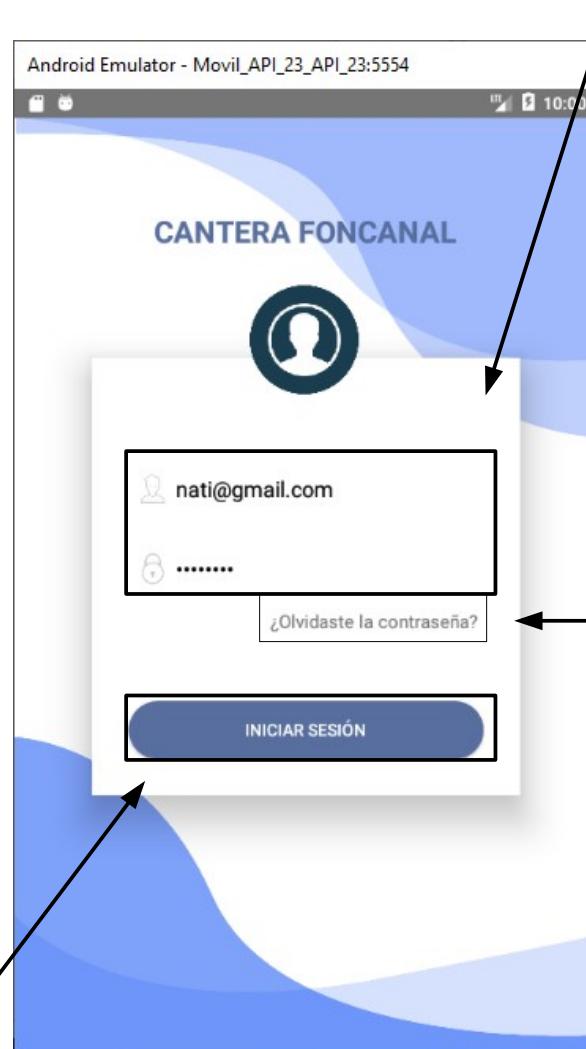
Finalmente para poner en funcionamiento la web, es necesario añadir las carpetas del proyecto dentro de la carpeta htdocs de xampp.

El proyecto y la base de datos serán aportados por la empresa.

10. Manual de usuario

- Manual de Usuario en Android

Para tener acceso en la app, antes tienes que haber sido dado de alta en la empresa.



Email y Contraseña:

Cuando eres dado de alta la empresa te creará una cuenta de acceso con un email y una contraseña, el email no podrá ser modificado pero la contraseña si.

¿Olvidaste la contraseña?

Podrás cambiar la contraseña por una nueva.

Iniciar Sesión:
Nos dará acceso a la app.

Una vez hayas iniciado sesión se abrirá una nueva ventana.

Menú Lateral Izquierdo:

Pulsando en el, se desplegará un menú lateral

Menú Superior Derecho:

Pulsando en el, se desplegarán pequeñas opciones de menú.



Control Horario

Botón que redirige al Control Horario.

Productos

Botón que redirige a la gestión de Productos.

Voladuras

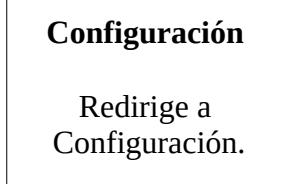
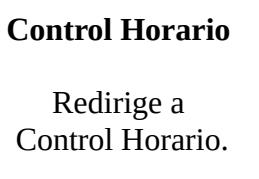
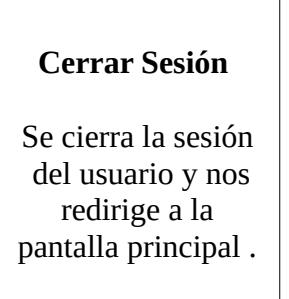
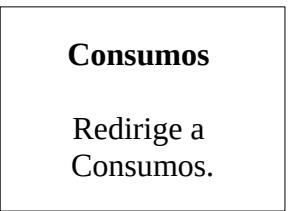
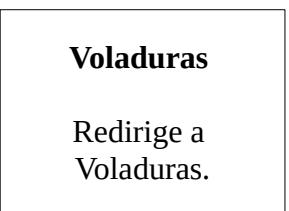
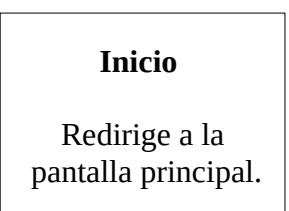
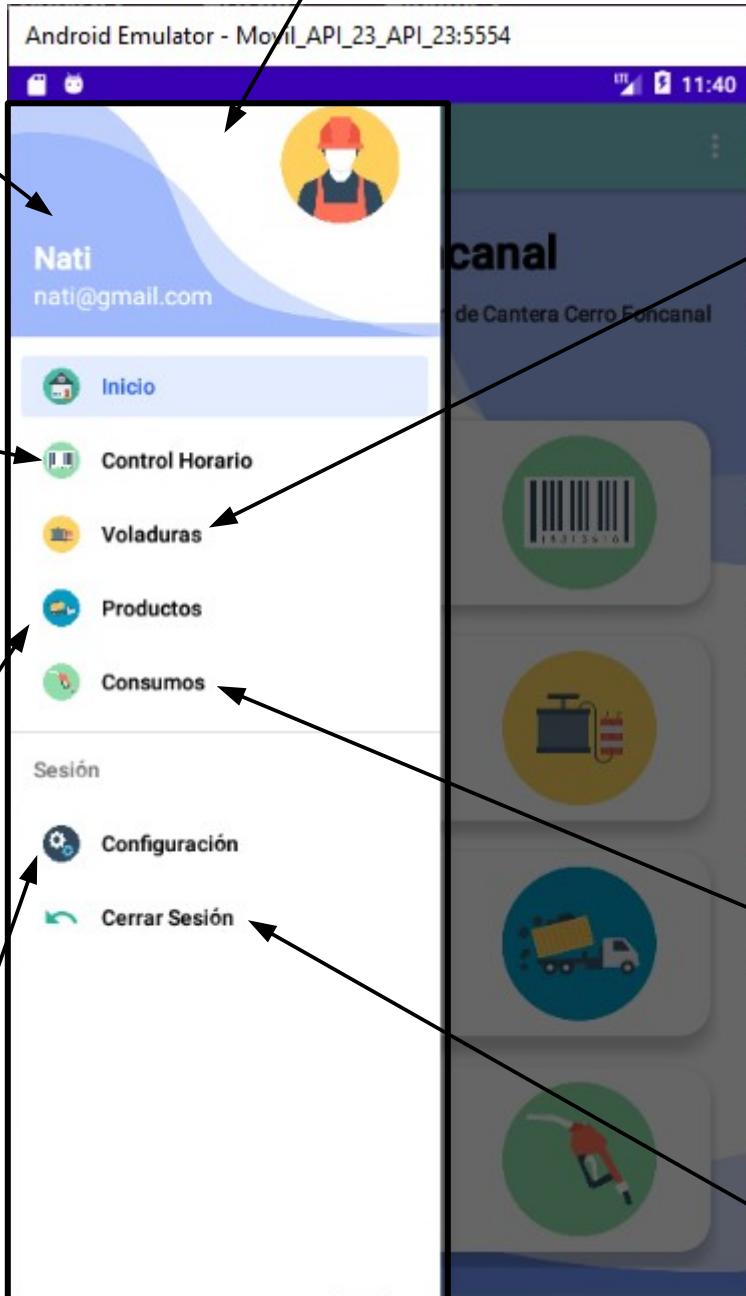
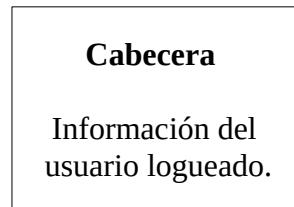
Botón que redirige a la gestión de Voladuras.

Consumos

Botón que redirige a la gestión de Consumos.

Si se pulsó , se abría un menú lateral.

Menú Lateral Izquierdo



Control Horario

Mensaje de Bienvenida

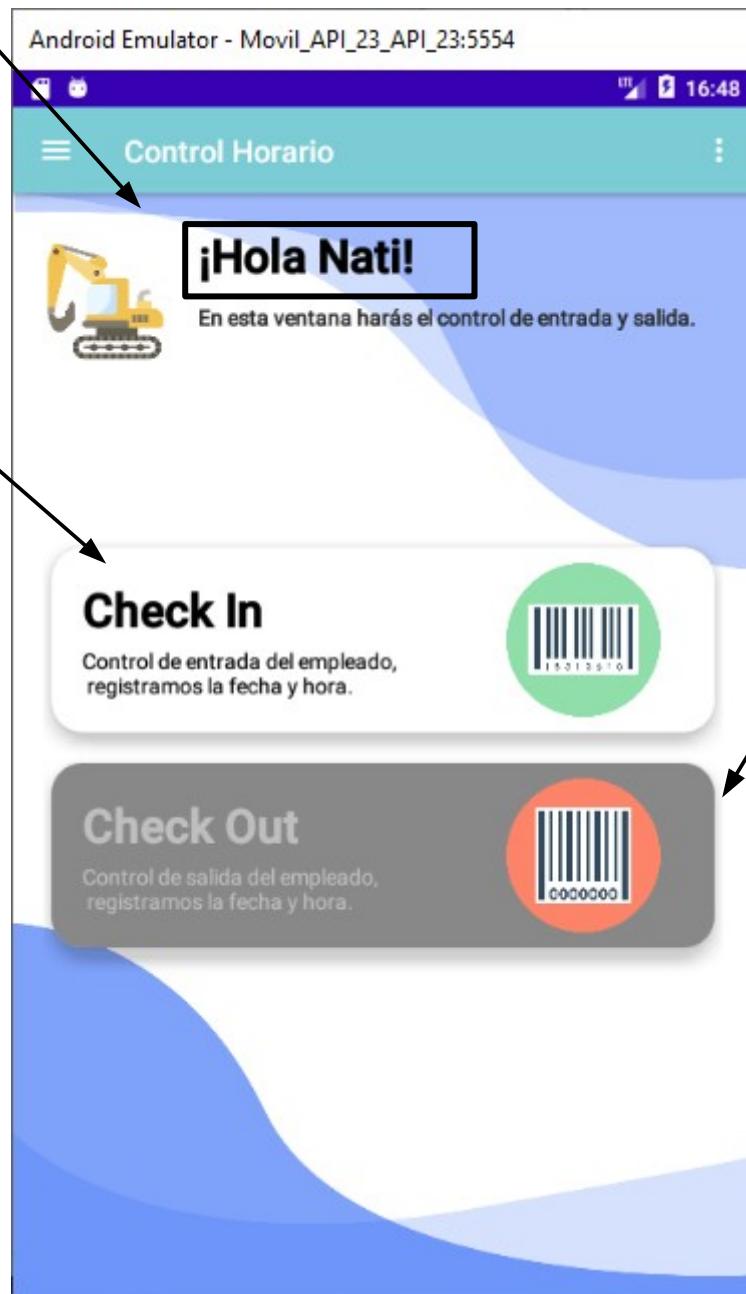
Check In

Botón que al ser pulsado realiza el check in diario de la hora de entrada.

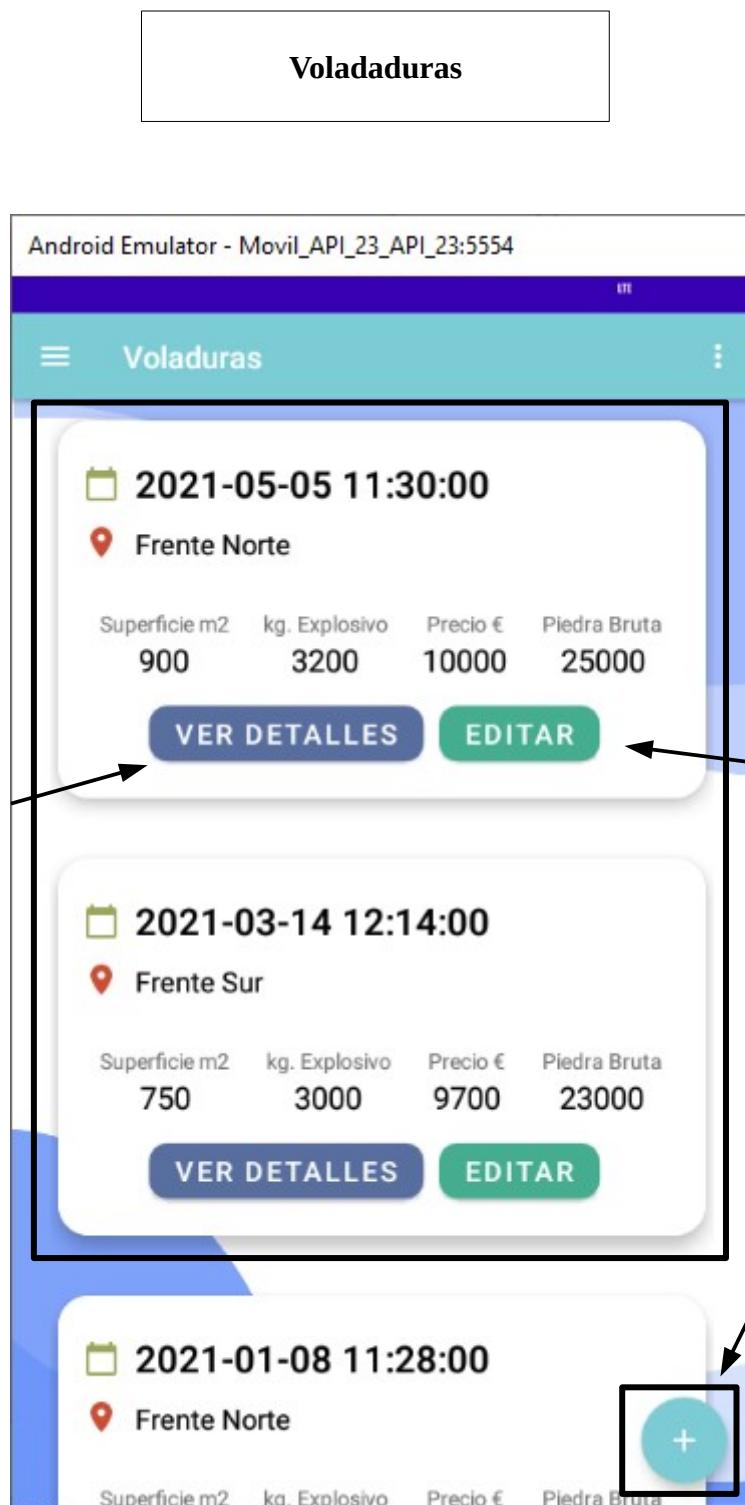
Check Out

Botón que al ser pulsado realiza el check out diario de la hora de salida.

(Estará habilitado cuando previamente se haya realizado el check in)



A la hora de **mostrar, editar, añadir o eliminar** los datos de una voladura se hacen igual que en productos, consumos y sus respectivos stock, por tanto sólo se explicará el ejemplo de voladura.



Listado Voladuras

Se muestran todas las Voladuras existentes.

Detalles

Botón que abrirá un diálogo mostrando todos los detalles de la Voladura pulsada.

Voladuras

Editar

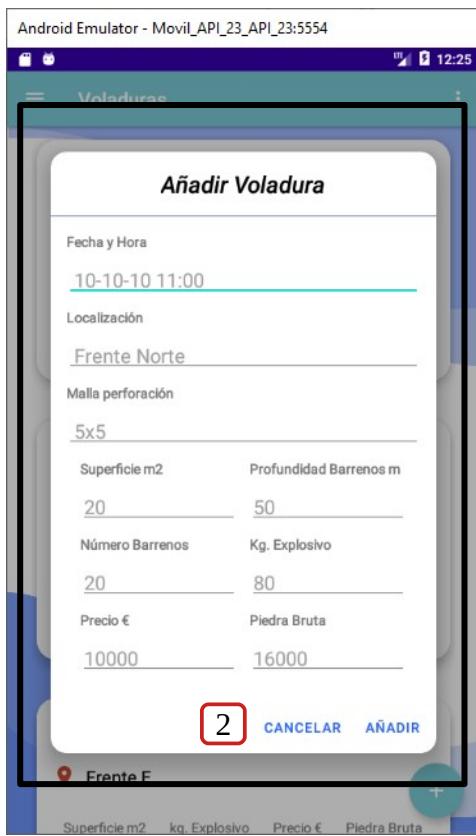
Botón que abrirá un diálogo mostrando todos los datos de la Voladura pulsada para ser editados.

Añadir

Botón que abrirá un diálogo para añadir una nueva Voladura.

Diálogos

A



B

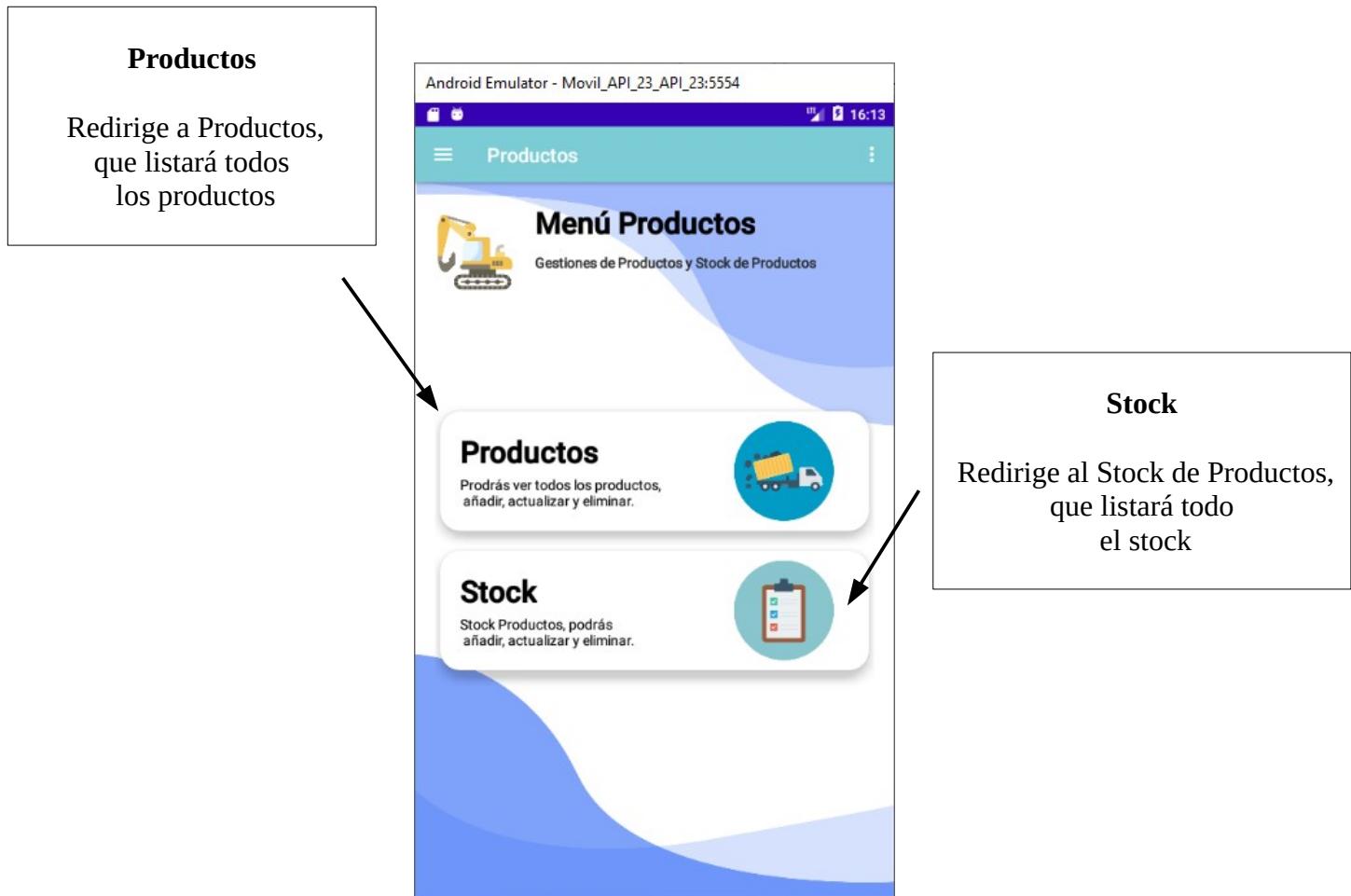


C

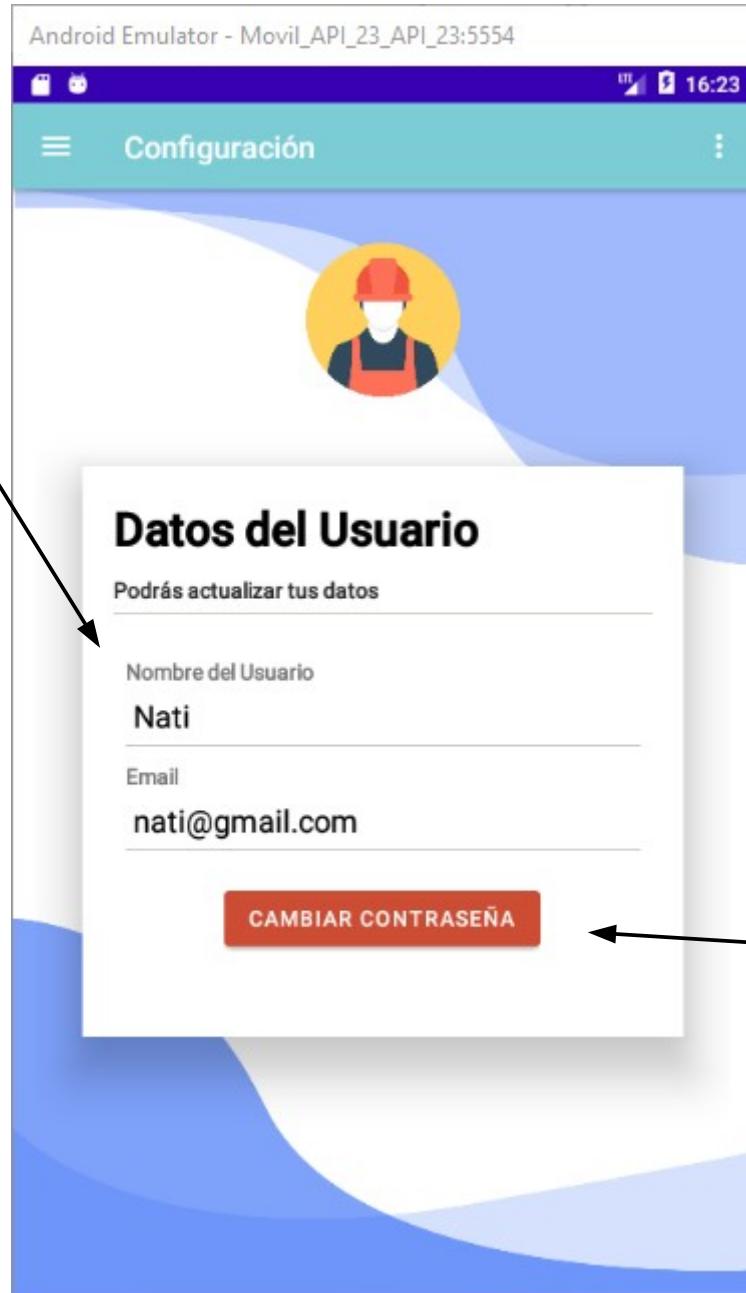


1. Los datos varían según la opción del menú elegida.
2. A → Diálogo Añadir → Añadir nueva voladura
 - **Añadir** → Añadirá una nueva voladura si los datos son correctos.
 - **Cancelar** → Cerrará el dialogo sin guardar.
3. B → Diálogo Editar → Editar voladura seleccionada
 - **Modificar** → Actualizará la voladura seleccionada si los datos son correctos.
 - **Cancelar** → Cerrará el dialogo sin guardar.
4. C → Diálogo Detalles → Detalles voladura seleccionada
 - **Eliminar** → Eliminará la voladura seleccionada.
 - **Aceptar** → Cerrará el dialogo.

En las opciones Productos y Consumos hay una ventana que no contiene voladuras, esto es para elegir los Productos / Consumos o el Stock.



Configuración



Datos Usuario

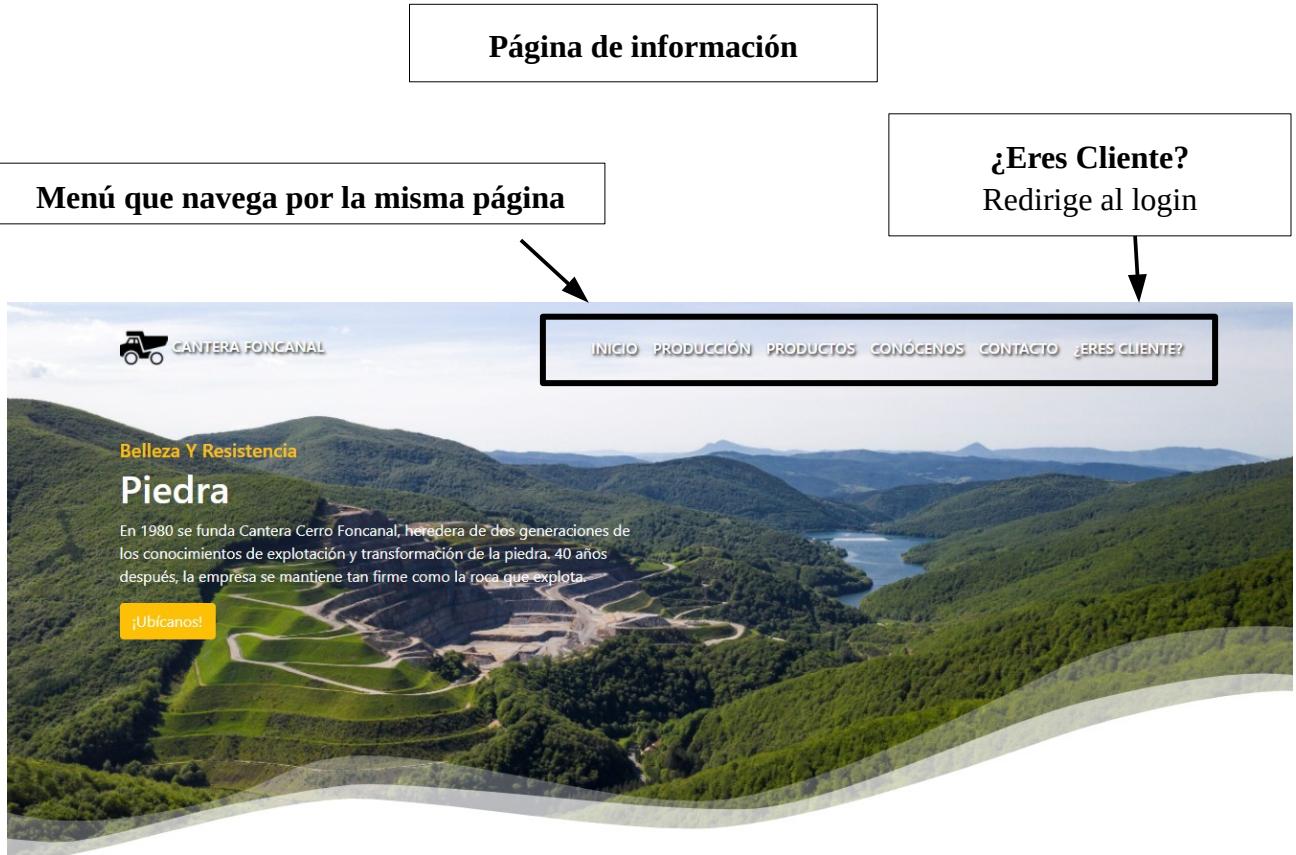
Nombre y email del usuario

Contraseña

Al pulsar el botón, aparecerá un diálogo, donde ingresaremos la contraseña actual, nueva y verificada.

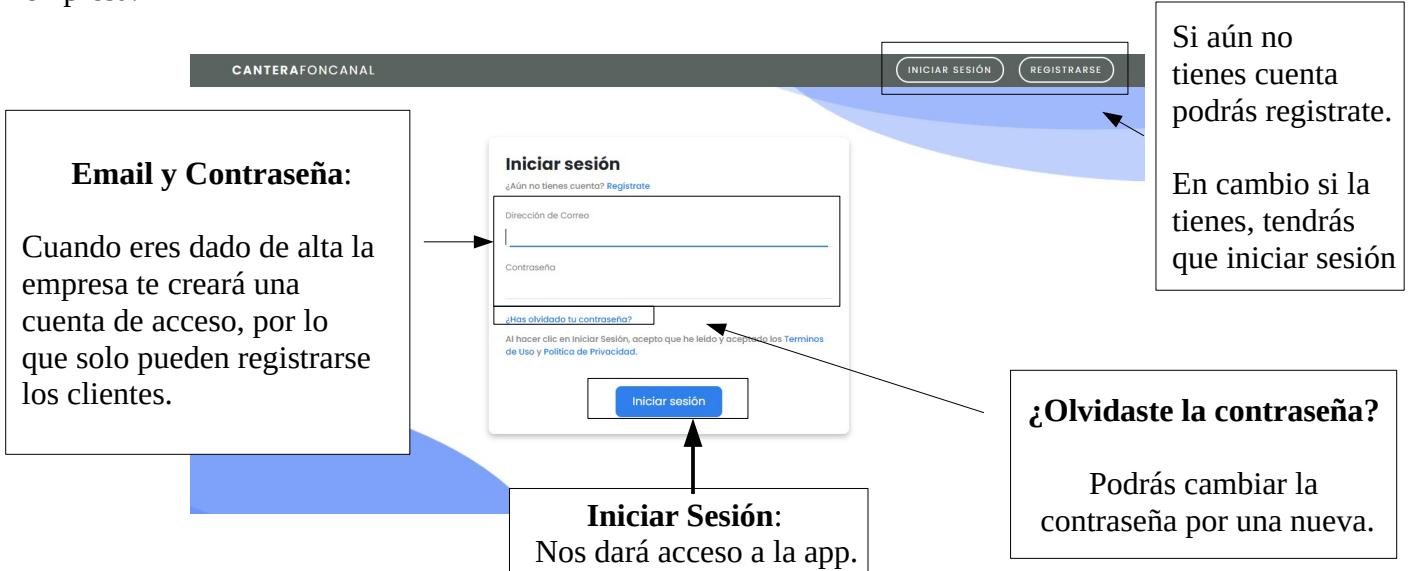
(Se cambiará si los datos introducidos son correctos)

- Manual de usuario para Web



Inicio de Sesión o Registrarse

Los únicos usuarios que pueden registrarse son los clientes, los empleados son registrados por la empresa.



La parte del cliente no ha sido desarrollada, pues nos hemos centrado más en el administrador y los empleados, en futuras actualizaciones podrá hacer uso completo de esta.

The screenshot shows a registration form titled "Registrarse". It includes fields for Nombre, Dirección de Correo, Contraseña, and Confirmar Contraseña. Below the fields is a checkbox for accepting terms and conditions, followed by a "Registrarse" button. At the top right are "INICIAR SESIÓN" and "REGISTRARSE" buttons. A callout box on the left says "Datos a Rellenar:" and the text "El Cliente tendrá que llenar todos los campos, pues si no es así, no podrá registrarse correctamente y tener acceso a la web." A callout box on the right says "Registrarse: Nos dará acceso a la web."

Datos a Rellenar:
El Cliente tendrá que llenar todos los campos, pues si no es así, no podrá registrarse correctamente y tener acceso a la web.

Registrarse:
Nos dará acceso a la web.

Esto sería lo primero que nos encontramos al iniciar sesión como administrador o empleado.

The screenshot shows the main page with a banner featuring a construction site and the text "BIENVENIDO A CANTERA CERRO FONCANAL". The top navigation bar includes links for "Inicio", "Departamentos", "Control Horario", and a user profile icon. A callout box on the left says "Menú:" and "Menú para navegar por la web". A callout box on the right says "Mensaje de bienvenida".

Página Principal / Inicio

Menú:
Menú para navegar por la web

Mensaje de bienvenida

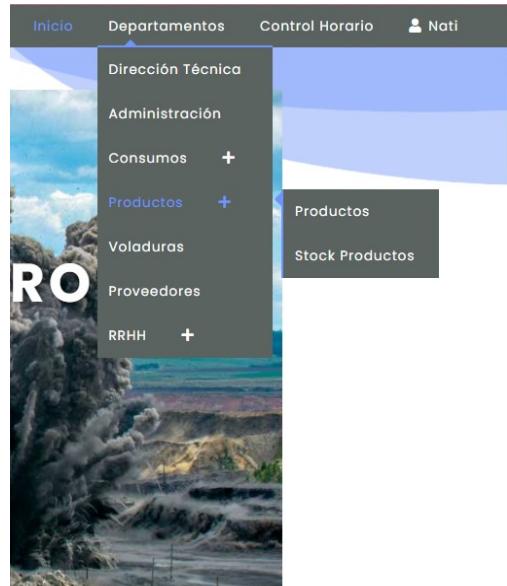
Opciones de Menú / Submenú

Todos estos submenús nos dirigen a una nueva ventana.

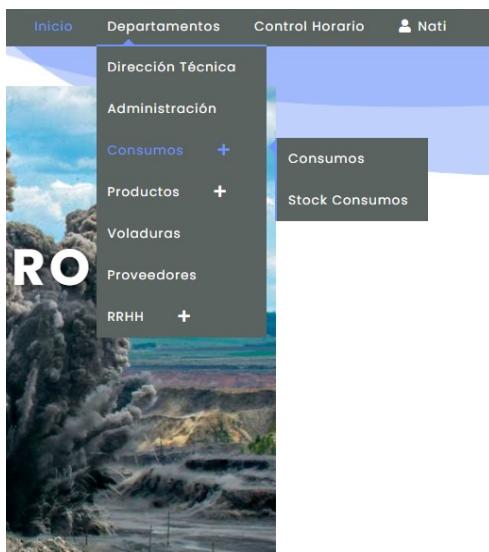
Recursos Humanos



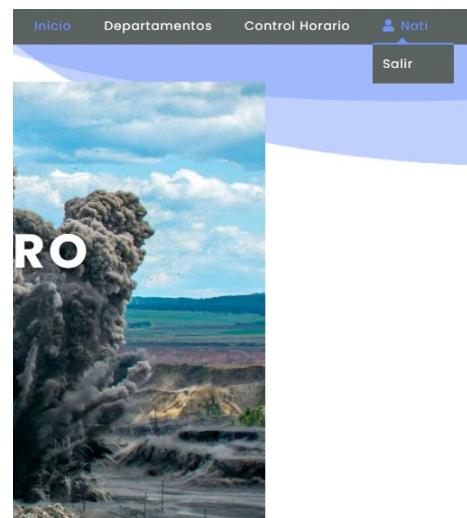
Productos



Consumos



Cerrar Sesión



Al salir, volveremos a la pantalla de información

Se mostrará sólo la tabla de voladuras, ya que todas las demás son similares (empleados, productos, consumos y sus respectivos stock)

Voladuras

Mostrar Registros
 La tabla mostrará el número de filas que elijamos, 5 – 10 - 20 o todos las filas.

Botones:
Verde → Excel
Rojo → PDF
Azul → Imprimir
 (sólo valores que se reflejan en la tabla)

Buscador
 Filtra por todas las columnas, poniendo el valor, buscará la fila que necesites.

CANTERAFONCANAL
Inicio / Voladuras
Inicio De

Mostrar 5 registros

Buscar:

Fecha Voladura ↑	Localización ↑↓	Superficie m ² ↑↓	Malla de Perforación m ↑↓	Profundidad Barrenos m ↑↓	Número Barrenos ↑↓	Kg Explosivo ↑↓	Precio € ↑↓	Piedra Bruta tm ↑↓	Acción ↑↓
2020-12-16 11:35:00	Frente Sur	750	4x3.5	19	56	3200	10000	25000	
2021-01-08 11:28:00	Frente Norte	900	3x3.5	16	62	3200	10000	25000	
2021-03-14 12:14:00	Frente Sur	750	3.5x3.5	18	58	3000	9700	23000	
2021-05-05 11:30:00	Frente Norte	900	3x3.5	16	62	3200	10000	25000	

Mostrando registros del 1 al 4 de un total de 4 registros

Anterior
1
Siguiente

Migas de pan
 Ayuda a la hora de navegar, para saber donde estamos situados

Anterior / Siguiente
 Movernos entre los registros

Añadir
 Añadir un nuevo empleado

Acciones
 Permite Editar, Mostrar y Eliminar la fila elegida.
 Si elegimos Editar, redirigirá a un formulario con los datos rellenos de la fila, si elegimos Mostrar, nos muestra una ficha de esta fila y si elegimos Eliminar se abrirá un cuadro de diálogo, para confirmar.

83

Acciones

El formulario añadir/registrar y de editar/modificar es completamente igual, excepto que el de editar ya nos autocompleta los campos, podemos cambiarlos si es lo que se quiere. (Se podrá añadir o editar si los campos introducidos son correctos)

Añadir una nueva voladura

Registrar Voladura

Creación nueva voladura.

Fecha Voladura	Localización	Superficie m2
dd/mm/aaaa --:--	<input type="text" value="Frente Norte"/> ▼	
Malla Perforación m	Profundidad Barrenos m	Número Barrenos
Kg. Explosivo	Precio €	Piedra Bruta tm

Registrar **Volver Atrás**

Modificar una voladura

Modificar

Modificar voladura.

Fecha Voladura	Localización	Superficie m2
02/05/2021 22:17	<input type="text" value="Frente Norte"/> ▼	10
Malla Perforación m	Profundidad Barrenos m	Número Barrenos
5x5	10	20
Kg. Explosivo	Precio €	Piedra Bruta tm
100	1000	2000

Modificar **Volver Atrás**

Volver Atrás
Como el nombre indica,
Volverá a la página anterior

Detalles / Ficha

Se reflejará todo sobre la fila de la voladura seleccionada.

The screenshot shows a mobile application interface. On the left, a dark green sidebar labeled "Inf. Voladura" contains the date "2021-05-02 22:17:00" and location "Frente Norte". The main area is titled "Ficha Técnica" and displays technical data in two columns:

Superficie m ²	10 m ²
Malla de Perforación	5x5 m
Profundidad Barrenos	10 m
Número Barrenos	20
KG. explosivo	100 kg
Precio	1000 €
Piedra Bruta	2000 tm

A "Volver atrás" button is located at the bottom of the main content area.

11.- Bibliografía

www.viaempresa.cat

www.mctabogados.com

www.ttandem.com

<https://developer.android.com/>

<https://laravel.com/>

<https://getbootstrap.com/>

<https://es.stackoverflow.com/questions/122796/cual-es-la-funci%C3%B3n-del-targetsdkversion>

<https://stackoverflow.com/questions/53902494/navigation-component-cannot-find-navcontroller>