Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERÍAS INDUSTRIALES

Grado en Ingeniería en Tecnologías Industriales

APLICACIÓN WEB PARA LA PLANIFICACIÓN Y CONTROL EN TIEMPO REAL DE LA PRODUCCIÓN

Autor:

Alonso Villanueva, Francisco Luis

Tutor:

Araúzo Araúzo, José Alberto.

Valladolid, septiembre de 2021.

Resumen

Se ha desarrollado una aplicación web, basada en TypeScript y SQL, que permite el control de la producción en tiempo real del taller de la empresa Isringhausen, situada en Valladolid. Gracias a ella cualquier empleado autorizado de la empresa podrá comprobar exactamente la productividad de cada trabajador en cualquier momento del turno, pudiendo ver esta información acompañada de estadísticas, bien por puestos o bien de forma general. Esta aplicación supone un avance hacia la transformación digital de la empresa, teniendo repercusión, aunque de distintas maneras, en prácticamente la totalidad de ésta.

Para su funcionamiento también se ha creado una base de datos, que no sólo surte de información al programa, sino que también almacena todos los registros que día a día se van generando. De esta forma, en cualquier momento se puede acceder a ella y comprobar datos pasados, con su consiguiente reducción de trabajo administrativo.

Palabras clave

Base de datos, digitalización, aplicación web, control de la producción, tiempo real.

Abstract

A web application has been developed, based in TypeScript and SQL, which allows the control in real time of the factory production of the Isringhausen company, located in Valladolid. Thanks to it, any authorised employee of the company will be able to check exactly the productivity of each worker at any time of the shift, being able to see this information accompanied by statistics, either by posts or in a general way. This application represents an advance towards the digital transformation of the company, having an impact, although in different ways, on practically all of it.

A database has also been created for its operation, which not only supplies information to the programme, but also stores all the records that are generated on a daily basis. In this way, it can be accessed at any time and past data can be checked, with a consequent reduction of administrative work.

Key words

Database, digitalisation, web application, productivity, real-time.

A mis padres, por haber hecho posible que llegara este día.

A mi familia y amigos, por haberme acompañado en todo el camino.

A Anabel, por su apoyo incondicional.

A mis compañeros de Isri, especialmente a Arkaitz y Guillermo, por su enorme implicación en el proyecto, y a Mª Ángeles por darme esta oportunidad.

Gracias.

Índice

Índice de ilustraciones

1. Introducción

1.1 Justificación del proyecto

El Trabajo de Fin de Grado que se va a realizar nace por la necesidad que tiene la empresa de optimizar el proceso de fabricación del taller. En la actualidad no existe ningún sistema que permita ver el momento en la producción en el que se encuentra cada trabajador. Con esta nueva herramienta, se podrá ver la cantidad exacta de elementos que ha fabricado cada uno en tiempo real, ya sea de forma detallada o general, y acompañada de estadísticas. Además, la recogida de datos se realiza de forma manual, con un Excel muy pesado, y en multitud de ocasiones, con errores. El hecho de que sea tan pesado hace que la tarea de creación y guardado de registros sea muy ardua, tardando entre veinte y treinta minutos en realizarse cada una de las tareas, con las molestias que ello conlleva. Con el nuevo sistema, este proceso se realizará de forma automática, recogiendo todos los datos que se deseen generados por las máquinas de cada puesto. Con ello se logrará un registro automático del trabajo de cada empleado del taller, pudiendo realizar con dicha información todos los estudios que se consideren necesarios para mejorar la productividad de la empresa. Para poder visualizar toda la información adecuadamente se creará una aplicación web, que facilitará el acceso y comprensión de todos los datos recogidos de las diferentes fuentes.

Esto beneficiará a la gerente y jefa de producción, que tendrá un mayor control de lo que suceda en el taller y los datos mucho más abundantes y accesibles. También servirá al encargado de cada turno, que como ya se ha mencionado antes, podrá comprobar la productividad de cada operario en cada instante, ayudándole así en la toma de decisiones de la jornada. También solucionará los problemas diarios que se tienen para crear y guardar registros, que llevan entre cuarenta minutos y una hora diaria, puesto que la nueva herramienta incluye un nuevo sistema de planificación. Con este sistema, la creación se hará instantáneamente, y el registro y guardado de forma automática. Por último también aportará cierta comodidad a los operarios, que ya no tendrán que anotar manualmente el trabajo que han realizado, ahorrándose así ese tiempo diariamente.

1.2 Objetivos

- Creación de una base de datos que albergue toda la información.
- Posibilidad de ver, en tiempo real, el trabajo realizado por cada empleado acompañado de estadísticas.
- Registro automático del trabajo diario de cada empleado.
- Facilitar y actualizar el sistema de planificación del trabajo diario.
- Crear un fácil acceso a dicho registro y a toda la información de la nueva base de datos.

1.3 Contenido

El presente documento comienza exponiendo en el capítulo *2*, la empresa y su actividad, seguida de cómo se va a integrar la aplicación en el sistema. Posteriormente se describirán en el capítulo *3* todas las herramientas utilizadas en el desarrollo del proyecto, y en el capítulo *4*, se explicará con detalle cómo se ha creado la aplicación y su funcionamiento. Finalmente se desarrollarán las conclusiones en el capítulo *5*.

2. Empresa e innovación

En este capítulo se hablará de la empresa y su funcionamiento interno, se dará una breve explicación del sistema que se va a implantar y se contarán las fases por las que ha pasado el autor del proyecto en su desarrollo.

2.1 La empresa

Isringhausen, también conocida como ISRI, es una empresa alemana dedicada a la fabricación de asientos, principalmente para autobuses, camiones y furgonetas. Ésta nació en 1919 en Bielefeld, ciudad al norte de Alemania, como una empresa fabricante de asientos de bicicleta. Aunque predomine su presencia en Europa, cuenta con plantas en los cinco continentes. En España tiene fábrica en Pamplona, la más grande del grupo Íbero, Valladolid y Madrid, además de otra en Portugal, perteneciente al mismo grupo. El proyecto que se va a presentar a continuación trata únicamente la de Valladolid. Esta planta está situada físicamente dentro de la fábrica de Iveco, empresa para la que aquí se produce la totalidad de los asientos.

2.2 Funcionamiento de la empresa

El taller está organizado en islas de trabajo independientes, de forma que cada operario puede controlar su ritmo de producción, siempre y cuando alcance la producción exigida. Dichas islas se dividen en preensamblado, tapizado y ensamblado final. Las primeras son las operaciones más sencillas y consisten en el montaje general de piezas, tales como suspensiones, guías o estructuras. A continuación iría el tapizado de los respaldos y banquetas, que en función del modelo necesitarán que previamente, en el preensamblado, se hayan producido ajustes lumbares o no. Posteriormente se realizaría el montaje final de los asientos, siendo éstos los puestos que más tiempo y formación requieren.

Tras realizar las debidas comprobaciones a cada asiento ya fabricado, éstos se disponen sobre unos rodillos, los cuales van a parar a una carretilla con vagones que los recoge y realiza viajes periódicos directamente a la cadena de Iveco, suministrando así el material que necesitan.

El suministro de materiales al taller se realiza de forma directa desde el almacén con una carretilla con vagones. Para los asientos biplaza trae en cada vagón el material exacto que se necesita para montar dos de ellos. Esto serían las fundas, espumas, cinturones de seguridad, etc., que tendrá que utilizar el operario. Las banquetas y los denominados *tavolinos* se toman directamente de las dos islas de producción que se encuentran al lado.

Los asientos individuales también reciben el suministro preparado para cada uno, pero a diferencia de los dobles, estos se hacen directamente en el propio taller. Estos lotes se preparan con materiales que provienen de los diferentes puestos de trabajo, es decir, del preensamblado y del tapizado, los cuales convergen en un mismo punto en el que se preparan dichos lotes.

Para saber cuáles se deben producir, ISRI tiene comunicación constante con Iveco a través de una plataforma informática, que informa con suficiente antelación de su producción más próxima, y por tanto, de lo que va a necesitar en las próximas horas. También existe un sistema que informa de las necesidades a medio plazo, para que ISRI pueda estimar los materiales que serán necesarios y pedirlos a los proveedores.

Todos los puestos tienen un sistema, aunque no para todas el mismo, que contiene un escáner que recoge información del proceso, tal como la fecha y hora, trabajador que realiza la operación, número de serie del asiento o pieza, etc. Estas máquinas en la mayoría de los casos son atornilladoras, representadas en rojo en la *Ilustración 1*. También recogen información el sistema de comprobación del correcto funcionamiento de los sensores sensibles al peso (máquinas de tapizado); el denominado "EOL", máquina situada en el último puesto del proceso de fabricación y que únicamente se dedica a revisar que todo esté conforme a las exigencias de Iveco; y los puestos para los que se ha implantado un escáner para poder recoger datos para este proyecto. De esta forma, todos los puestos recogen información suficiente como para poder hacer un seguimiento del trabajo que se ha realizado en cada uno, por quién y cuándo.

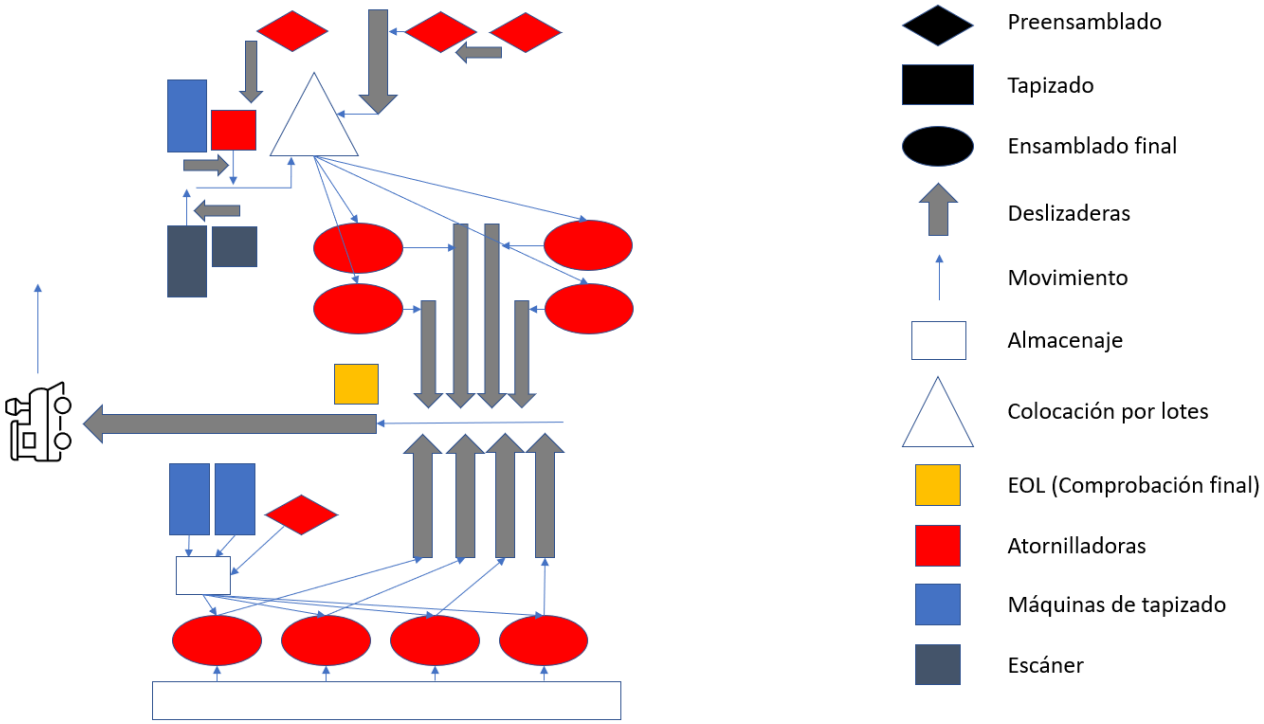


Ilustración 1. Mapa del taller

2.3 Funcionamiento del sistema

En la *Ilustración 2* se muestra el funcionamiento del sistema y dónde se guardan los datos. Actualmente existe un servidor que recoge todos los datos generados por las atornilladoras y el EOL en formato SQL. Además contiene el programa creado en este proyecto. Como ya se ha explicado anteriormente y mostrado en la *Ilustración 1*, las atornilladoras cubren la mayoría de los puestos. Las dos máquinas de tapizado y los puestos de respaldos y lumbares tienen un sistema aparte que guarda la información localmente, por lo que se hace la conexión directamente a ellas. Es por ello por lo que se representan separadas.

Posteriormente se realiza un cribado de la información que se recoge, ya que se generan muchos datos, pero sólo unos pocos son interesantes para nuestro proyecto. Con este cribado se crea un nuevo SQL, al que se conectará tanto el *api rest* como la aplicación, que será donde se mostrará toda la información a quien lo desee y tenga acceso.

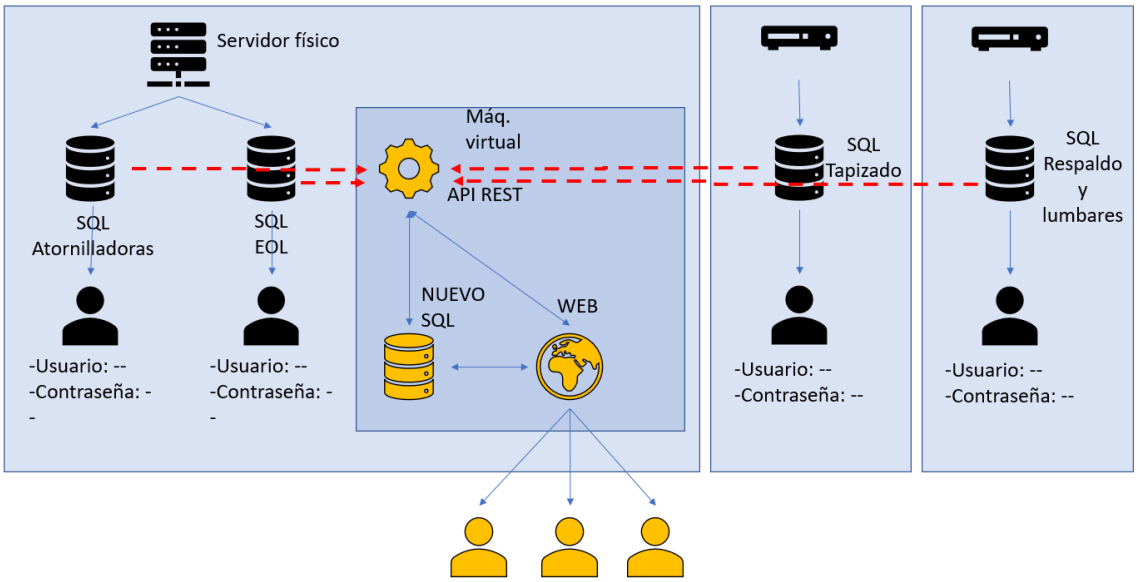


Ilustración 2. Funcionamiento del sistema

2.4 Fases

2.4.1 Conocimiento general de la empresa

El primer trabajo encomendado al llegar a la empresa nada tuvo que ver con el proyecto que se está presentando. Éste fue actualizar las instrucciones de trabajo de la empresa. Estas instrucciones recogen detalladamente cada uno de los pasos que se siguen en los puestos de trabajo del taller, de forma que podría realizarse cualquiera de los procesos únicamente siguiendo las instrucciones.

Esta tarea supuso principalmente cambiar algunos pasos que ya no se realizaban de esa forma, añadir algunos nuevos, ya que los modelos de los asientos cambian con el tiempo, y sobre todo, actualizar las referencias de las piezas. Para ello se comprueba puesto por puesto observando cómo trabaja cada operario y comparándolo con las instrucciones que se habían recibido. Cuando se encontraba alguna diferencia, se hablaba con el encargado y se debatía si era necesario o no cambiar la instrucción. Cuando sí que lo era, o bien se eliminaban pasos, o bien se añadían, acompañados de una descripción del proceso y fotos.

Las referencias se comprueban mediante web interna de la empresa, buscando los planos del conjunto en el que va integrada, y a partir de ahí, buscar nuevos planos de más subconjuntos hasta encontrar la pieza concreta. Por ejemplo, si se quiere encontrar el número de referencia del amortiguador de la suspensión, se tiene que buscar el plano completo de un asiento que lleve dicha pieza, buscar el subconjunto de la suspensión, y en él, el amortiguador con su referencia. Si no se encontrara, se tendría que seguir indagando desde ahí hacia partes cada vez más concretas.

Todo esto sirvió principalmente para conocer todo el funcionamiento de la empresa, cómo se trabaja y todo el proceso de fabricación de los asientos, desde las partes más pequeñas hasta

el conjunto acabado y revisado. Por supuesto, también valió como método de integración, ya que obliga a hablar con una gran parte de los trabajadores que expliquen cada proceso.

2.4.2 Formación

Tras acabar esa primera parte comenzó la formación en el trabajo específico que debía hacer. Se dieron pautas más detalladas de cómo debería quedar el trabajo final y se habló de algunas de las herramientas que tendría que utilizar. Tras ello comenzó un aprendizaje autónomo sobre el programa Microsoft SQL Server Management Studio 18 (de ahora en adelante se hará referencia a él como MSSQL) y su lenguaje.

Como ya se ha comentado antes, ISRI tiene una planta mucho mayor en Pamplona, donde trabaja un informático y un ingeniero que serán fundamentales para el desarrollo del proyecto, estando enormemente implicadas en él. Se programó una estancia de una semana con ellos para aprender programación más avanzada, principalmente TypeScript, y a ultimar la planificación general del proyecto, es decir, hacerla mucho más detallada, paso a paso y con plazos para cada parte.

Semanas después se volvería para ampliar mi formación en otra fase del proyecto.

2.4.3 Creación de la base de datos

En primer lugar, se crea el soporte donde se almacenará toda la información. Esto son tablas en MSSQL, que además estarán relacionadas entre sí. Su carácter es variado, hay algunas que apenas variarán en el tiempo, como puede ser la tabla de “Encargados” o la de “Trabajadores”. En ellas se especifica el nombre de la persona, un número de identificación y turno al que pertenecen. Otras estarán registrando datos de forma constante, como por ejemplo la de “Registros”, donde se almacenará el trabajo diario de cada operario.

Estas tablas, junto a lo que será programado posteriormente en Visual Studio Code, serán la base de todo el trabajo. Dicha programación básicamente lo que hará será acceder a ellas para leerlas, actualizarlas, añadir o eliminar datos, para después mostrarlos en la página que se creará.

2.4.4 Creación de una *REST-API*

Esta parte es el *backend* del proyecto. Aquí se creará la *REST*-API con Visual Studio Code. Ésta será la encargada de acceder y manipular la información de la base de datos creada. Tras conectar la *REST-API* con MSSQL, se crean las rutinas para la utilización de los datos siguiendo las siguientes asociaciones: *POST* se utiliza para crear nuevos datos, *GET* para leerlos, *PUT* para actualizar dichos datos y *DELETE* para eliminarlos.

Al estar conectados a MSSQL, cuando se ejecute una de estas asociaciones se mandará el código correspondiente a dicho programa. Por ejemplo, cuando se ejecuta un *GET*, como ya se ha determinado anteriormente, lo que el usuario está buscando es acceder a la base de datos para leerla. Para hacer esto en MSSQL se realiza un *SELECT*, por lo que se enviará al programa dicho código (P. ej.: *SELECT * FROM Encargados*) y se obtendrá toda la información de almacenada que se especifique.

Esto se incluirá en unos archivos que se denominarán *Controllers*, donde estarán los controladores y habrá uno para cada tabla que hemos creado. Éstos irán vinculados con otros

que denominaremos *Routes*, donde estarán almacenadas las rutas, y para los que también habrá un archivo por tabla.

2.4.5 Diseño web del acceso a la base de datos

Parte correspondiente al *frontend* del trabajo. Desde este acceso los usuarios interactuarán con la base de datos. En esta fase se diseña la aplicación web al completo y se realizan los documentos de tipo .ts, .html, y .css. Los primeros son TypeScript, en ellos se crean las funciones que se utilizan para que la aplicación funcione. Los segundos tratan el diseño, la estructura de la página, y los terceros dan el formato a lo establecido en el documento HTML.

3. Herramientas

El presente proyecto se realizará utilizando principalmente los programas informáticos Microsoft SQL Server Management Studio 18 (MSSQL) para la gestión de la base de datos, Visual Studio Code como editor de código para el desarrollo del programa, Node.js como plataforma para el desarrollo del programa y Postman como intermediario entre el *backend* y *frontend* para comprobar el funcionamiento de la primera parte. Se utilizará el método Scrum de trabajo a través del programa Jira para mejorar la coordinación entre los integrantes. También se usará Git para realizar todas las copias de seguridad y compartir código entre los miembros, entre otras cosas. Por último, aunque actualmente no influya en el funcionamiento del programa, sí que se ha comenzado a trabajar con Socket.io de cara a líneas futuras, para la actualización periódica y automática de los datos.

3.1 SQL

El lenguaje SQL (del inglés "*Structured Query Language*", en español "*Lenguaje de Consulta Estructurada") "es un lenguaje de computación para trabajar con conjuntos de datos y las relaciones entre ellos*"(Microsoft, s. f.). A través de MSSQL se recogerán y manipularán los datos del servidor que constantemente están generando las máquinas que tienen los operarios en el taller, tales como las atornilladoras o máquinas de comprobación del correcto funcionamiento de los asientos, de las que se hablará más adelante. También lo utilizaremos para crear nuestras propias bases de datos con la información que sea necesaria.

SQL está basado en el modelo relacional de bases de datos, siendo la relación lo más importante del modelo. Ésta está formada por un conjunto de filas y columnas que forman una tabla de una entidad de datos relacionados. Una entidad es aquello sobre lo que se recogen los datos. Las características que tiene se clasifican en atributos o columnas. Por ejemplo, en una tabla de los trabajadores de una empresa, los trabajadores serían la entidad y los atributos serían los datos que se recogen sobre ellos, tales como su nombre o fecha de nacimiento. La información se almacena en filas, recogiendo cada una un registro de los datos.

Otra característica muy importante es que debe escogerse un identificador único o clave principal. Esto es un atributo o conjunto de atributos que sirven para diferenciar una fila de cualquier otra de forma inequívoca.

Este modelo brevemente explicado nació en 1970, gracias al científico informático inglés Edgar Frank Codd, trabajador de IBM, que publicó su libro "*A Relational Model of data for Large Shared Data Banks*", en español, "Un modelo relacional de datos para grandes bancos de datos compartidos". Posteriormente IBM comenzó a desarrollar un lenguaje y un sistema de base de datos que aplicara este modelo, dando lugar a SEQUEL, del inglés "*Structured English Query Language*". Posteriormente habría problemas legales con este nombre por pertenecer ya a otra empresa, por lo que se cambió a SQL. A raíz de ello, otras empresas se lanzaron a desarrollar sus propios programas basados en SQL, haciendo así que alcanzara una gran popularidad.

En 1982, ANSI (*American National Standards Institute*), comenzó la búsqueda para la definición de un lenguaje basado en el modelo relacional de bases de datos. En 1986, ANSI publicó el primer estándar de SQL y en 1987 fue adoptado por ISO (*Internacional Organization for Standardization*). Dicho estándar ha sido actualizado en 1989, 1992, 2003 y 2006. (Oppel & Sheldon, 2010)

3.2 Visual Studio Code

Visual Studio Code es un editor de código capaz de funcionar con múltiples lenguajes de programación (Visual Studio, s. f.). En nuestro caso se trabajará principalmente con TypeScript, un superconjunto de *JavaScript* que ofrece ciertas ventajas, tales como tener sintaxis adicional para una mejor integración con el editor, en este caso VS Code; detectar los errores más rápido, haciéndolo cuando se compila el archivo en lugar de cuando se está utilizando ya la aplicación; convertirse en código *JavaScript*, ejecutándose en cualquier lugar donde lo haga JavaScript; entender *JavaScript* y utilizar la inferencia de tipos para ofrecer herramientas sin código adicional. (Comunidad TypeScript, s. f.)

A través de este programa y los lenguajes mencionados se desarrollará la aplicación web a la que accederán los empleados de la empresa.

Anunciado el 29 de abril de 2015, este programa ha sido desarrollado por Microsoft para Windows, macOs y Linux (Montgomery, 2015). Algunas de sus grandes ventajas son:

- Contiene *IntelliSense*, una ayuda de autocompletado que proporciona terminaciones inteligentes. (Microsoft, 2018)
- El editor de código tiene multitud de características que permiten trabajar de forma mucho más cómoda. Entre las muchas que hay destacan la posibilidad de ver otras partes del código sin tener que ir a ellas, por ejemplo, pulsando sobre una función puede verse dónde ha sido declarada y el contexto que la rodea sin tener que desplazarte hasta ella; la barra que VS Code denomina como *breadcrumbs*, una barra que muestra constantemente la posición en la que se encuentra el desarrollador y que permite navegar rápidamente entre los archivos; sugerencias para solucionar errores detectados; detección del comienzo y final de paréntesis y un largo etcétera de características. (Visual Studio, 2021)
- Tiene los comandos de Git integrados. Más detalles sobre Git en *3.6*.
- Es ampliable y personalizable. Contiene innumerables extensiones que permiten conectarse a servicios y obtener herramientas adicionales, además de agregar nuevos idiomas y temas, entre otras cosas. El método de ejecución de estas extensiones hace que no se ralentice el editor por ellas.
- Es gratuito.

3.3 Node.js

Node.js es "*un entorno de ejecución de JavaScript orientado a eventos asíncronos, Node.js está diseñado para crear aplicaciones network escalables"* (Node.js, s. f.-a), es decir, es una plataforma para el desarrollo de aplicaciones *JavaScript*. Node.js ejecuta el motor JavaScript V8, el núcleo de Google Chrome, fuera del navegador. Esto permite que Node.js sea muy eficaz.

Node.js tiene una ventaja única y es uno de los motivos por los que se ha seleccionado este programa para el presente trabajo, y es que, si ya se sabe desarrollar el *frontend* con *JavaScript*, Node.js permite escribir el código del lado del servidor además del código del lado del cliente sin la necesidad de aprender un idioma completamente diferente.

El programa nació en el año 2009. *JavaScript* fue creado por *Netscape* para su navegador, no obstante, éste no tuvo mucho éxito. *JavaScript* se popularizó años después, con la introducción de Node.js. Uno de los motivos por los que Node.js ha triunfado es el momento

en el que se construyó, estando en el lugar y momento adecuados. Pocos años antes, *JavaScript* comenzó a tomarse como un lenguaje de programación más serio gracias a las más modernas aplicaciones "Web 2.0", como por ejemplo Flickr y Gmail. Sus motores mejoraron considerablemente para hacer que se ejecute más rápido y Node.js hizo lo propio con el suyo debido a la competencia. Sin duda esto ayudó a su crecimiento, pero por supuesto este no es el único motivo, ya que Introduce una gran cantidad de ideas y enfoques innovadores para el desarrollo del lado del servidor de *JavaScript* que ya han ayudado a muchos desarrolladores. (Node.js, s. f.-b)

3.4 Postman

Postman es "*una plataforma de colaboración para el desarrollo de API*" (Postman, s. f.). Como su propia definición indica, se utilizará durante el desarrollo de la API como método de comprobación de que todo funciona como se desea. Su procedimiento de uso en este proyecto es el siguiente, se introduce la ruta definida en el programa indicando la acción que se quiera realizar, de tipo *GET*, *POST*, *PUT* o *DELETE*. A continuación Postman ejecuta la orden programada y muestra por pantalla en formato .json la respuesta recibida. Su funcionamiento puede comprobarse de forma detallada en el capítulo *4.2.4*.

El programa nació en 2012, cuando un grupo de programadores indios creó el programa para solucionar su propio problema. Les resultaba demasiado tedioso hacer una simple llamada a la API y comprobar la respuesta para poder probarla y depurarla. Para ello crearon Postman, que hace exactamente eso, llamar a la API y mostrar la respuesta. Pronto decidieron subirlo a la *Chrome Web Store*, donde tuvo éxito y despegó. Con el tiempo ha ido evolucionando y creciente, hasta el punto de que hoy en día ya cuenta con 17.000.000 de desarrolladores utilizando el programa. (Postman, 2021)

3.5 Jira

Jira, es una herramienta en línea para administrar de forma eficiente el trabajo en equipo, siguiendo la metodología Scrum. Según su propia página web, se define como "*un conjunto de soluciones ágiles de gestión del trabajo que impulsa la colaboración entre todos los equipos, desde el concepto hasta el cliente. Ayuda a los equipos a planificar, asignar, rastrear, informar y administrar el trabajo y reúne a los equipos para todo, desde el desarrollo ágil de software y la atención al cliente hasta la puesta en marcha y las empresas*" (Atlassian, s. f.-b).

La metodología Scrum es un marco de gestión ágil de proyectos que usan los equipos para desarrollar, ofrecer y sostener productos complejos. Este sistema está muy extendido y puede utilizarse en cualquier ámbito de trabajo, pero donde más implantado está es entre los desarrolladores e ingeniería de software.

Una de las principales características del método Scrum es su capacidad para integrar un trabajo en iteraciones más pequeñas denominadas *sprints*. Consiste en dividir un gran proyecto en partes mucho más pequeñas que durarán entre dos y cuatro semanas como máximo, de forma que permite una mejor organización y tener objetivos claros marcados durante todo el proceso. (Atlassian, s. f.-c)

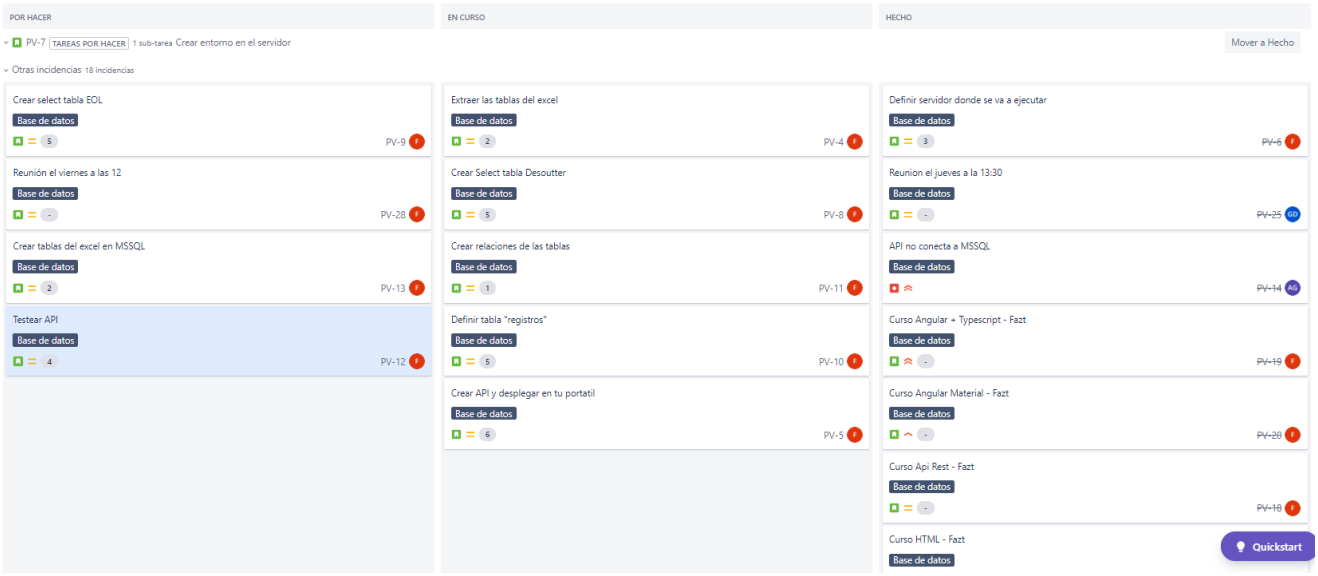


Ilustración 3. Tareas de un sprint en Jira

En este proyecto se ha utilizado para programar los pasos que debían darse desde el comienzo del proyecto hasta el final, añadiendo tareas e incidencias que surgían y asignando quién debe resolverlas, tal y como se ve en el ejemplo añadido en la *Ilustración 3*. Esto es muy importante ya que, al dividir un gran proyecto en pequeños objetivos, se consigue avanzar teniendo claras las ideas y acercándose siempre al objetivo final. Además, también se ha utilizado para asignar plazos a cada fase para finalizarlo en el tiempo establecido, como se muestra en la *Ilustración 4*. En definitiva, se ha realizado la programación completa de la aplicación, añadiendo cada paso que debía darse para cumplir cada uno de los objetivos.

Ilustración 4. Fases del proyecto

El programa Jira fue lanzado en el año 2002 por dos amigos de tan sólo 22 años. Desde entonces el programa no ha parado de crecer, contando ahora mismo con 5.000 empleados, oficinas en 7 países distintos y más de 180.000 clientes. También cuenta con una fundación, a través de la cual dona el 1% del capital social, el producto, las ganancias y el tiempo de los empleados a causas benéficas. (Atlassian, s. f.-a)

3.6 Git

Git es "*un sistema de control de versiones distribuido de código abierto y gratuito diseñado para manejar todo, desde proyectos pequeños a muy grandes, con velocidad y eficiencia*" (Git, s. f.-b).

Sus principales características, entre otras, son (Git, s. f.-a):

- Su capacidad de ramificación y fusión. Permite tener múltiples ramas locales que pueden ser completamente independientes entre sí. Esto implica que se pueden crear pequeñas versiones, ya sean, por ejemplo, de prueba o de trabajo diario que se pueden crear, combinar o eliminarse en cuestión de segundos.
- Es muy rápido. El hecho de que casi todas las operaciones se realicen de forma local da una gran ventaja frente a los sistemas centralizados que conectan con un servidor constantemente.
- Está distribuido. Esto significa que cada usuario tiene una copia de seguridad completa del servidor principal, por lo que ante un fallo o daño de éste, cualquiera de estas copias podría reemplazarlo.
- Es gratuito.

En este proyecto se ha utilizado principalmente para guardar copias de seguridad, acceder a ellas para comparar rápidamente distintas versiones del programa y para compartir código entre miembros del proyecto de forma instantánea.

El programa surgió en 2005. El kernel de Linux (el elemento principal del sistema operativo) es un proyecto de software de código abierto con un alcance bastante amplio. La comunidad que lo desarrollaba utilizaba un DVCS (control distribuido de versiones) que pertenece a BitKeeper. Cuando la relación entre dicha comunidad y BitKeeper se rompió, ésta comenzó a desarrollar su propia herramienta, dando lugar así a Git. (Git, s. f.-c)

3.7 Socket.io

Socket.io es "*una biblioteca que permite la comunicación en tiempo real, bidireccional y basada en eventos entre el navegador y el servidor*"(Socket.IO, 2021).

A pesar de que ya se ha probado con éxito en ciertos elementos, aún no se ha instaurado de forma completa y no influye en el desarrollo del programa. No obstante, se añade en este capítulo por ser el siguiente punto sobre el que se va a trabajar en la próxima versión.

Se utilizará para que periódicamente la aplicación obtenga todos los datos nuevos y se actualice, sin necesidad de presionar ningún botón para obtener la información. De esta forma se lograría un ahorro de tiempo aún mayor para todos los que revisen la aplicación de forma habitual durante la jornada, ya que lo que se mostrará por pantalla estará siempre actualizado.

4. Desarrollo

En el presente capítulo se va a explicar el desarrollo con detalle de todo el programa. Se comenzará explicando la base de datos, cada una de sus tablas y la información que se guarda en ellas. Posteriormente se explicará la creación de la *REST-API*, es decir, del *backend*. A continuación se darán unas nociones acerca de Angular Material, plataforma de desarrollo para el *frontend*, del que se habla seguidamente, explicando la aplicación completa página a página. A pesar de incluir algo de código, todo el restante se aporta en los anejos.

4.1 Creación de la base de datos

Esto consiste en la creación de las tablas en MSSQL. Estas serán:

1. *Encargados*.
2. *Trabajadores*.
3. *Clasificación de puestos*.
4. *Puestos*.
5. *Puestos físicos*.
6. *Experiencia*.
7. *Índice de las incidencias*.
8. *Incidencias*.
9. *Referencias de los asientos*.
10. *Planificación*.
11. *Registros*.
12. *Registros conjuntos*.

No todas las tablas sufrirán cambios con la misma frecuencia. Mientras que las tres últimas los tendrán diariamente de forma constante, las nueve primeras serán mucho más constantes en el tiempo. En la posterior explicación de cada una de ellas se muestran ilustraciones de las tablas, muchas de ellas han sido censuradas por motivos de privacidad de la empresa y sus trabajadores.

Todas tienen una columna con una clave principal (*primary key*) de tipo entero (*INT*). Su función es darle un valor único a cada fila, de forma que se pueda hacer referencia a la que se desee sin confusiones. Sólo puede haber una clave principal en cada tabla. Para relacionar las tablas entre ellas se utilizan claves externas (*foreign key*), que permiten relacionar una columna cualquiera de una tabla con la principal de otra. Por ejemplo, en la tabla “trabajadores” la clave principal es “id_trabajador”. Cuando “registros” indica a qué trabajador se refiere lo hace a través de este valor, haciendo en esta otra la función de clave externa.

A continuación, en la *Ilustración 5*, se muestra un diagrama de la base de datos donde se ven todas las tablas con los datos que almacenan y las relaciones que hay entre ellas.

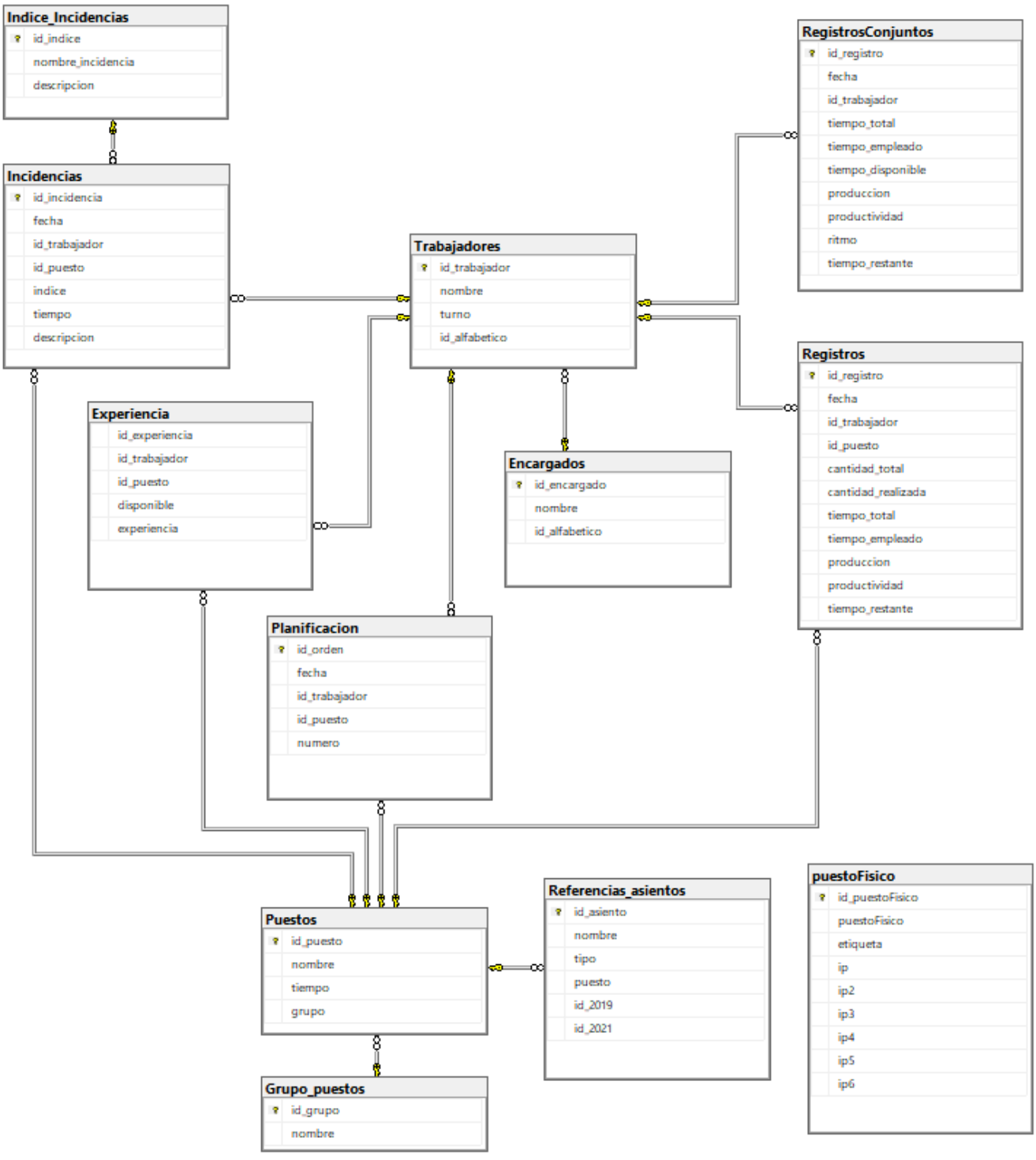


Ilustración 5. Diagrama de la base de datos

4.1.1 Encargados

Tabla muy sencilla que contiene una lista con el encargado de cada turno. Almacena su nombre y el *id* interno que se utiliza. Este último lo tiene cada empleado del taller impreso en una etiqueta y se escanea cada vez que se realiza una operación. De esta forma queda registrado en una base de datos interna cada acción realizada. Aparece también en la tabla de "trabajadores". Esta tabla se utiliza para relacionar a cada trabajador con su turno y encargado.

	id_encargado	nombre	id_alfabetico
1	1	[redacted]	[redacted]
2	2	[redacted]	[redacted]

Ilustración 6. Tabla de encargados

4.1.2 Trabajadores

Muy similar a la anterior, pero aquí se guardan los operarios del taller. Contiene su nombre e *id*, además del turno al que pertenecen. El turno es el *id* del encargado que lo dirige. Sirve para que las tablas posteriores, en caso de que lo necesiten, puedan hacer referencia a un trabajador. Éstas son las de incidencias, experiencia, planificación y registros.

	id_trabajador	nombre	turno	id_alfabetico
1	1	[redacted]	1	[redacted]
2	2	[redacted]	1	[redacted]
3	3	[redacted]	1	[redacted]
4	4	[redacted]	1	[redacted]
5	5	[redacted]	1	[redacted]
6	6	[redacted]	1	[redacted]
7	7	[redacted]	1	[redacted]
8	8	[redacted]	1	[redacted]
9	9	[redacted]	1	[redacted]
10	10	[redacted]	1	[redacted]
11	11	[redacted]	1	[redacted]
12	12	[redacted]	1	[redacted]
13	13	[redacted]	1	[redacted]
14	14	[redacted]	1	[redacted]
15	15	[redacted]	1	[redacted]
16	16	[redacted]	2	[redacted]
17	17	[redacted]	2	[redacted]

Ilustración 7. Tabla de trabajadores

4.1.3 Clasificación de puestos

Una tabla muy sencilla que recoge los tipos de puesto que tienen establecidos. "Puestos" está vinculada a ella, sabiendo así a qué grupo pertenecen.

	id_grupo	nombre
1	1	Pre-ensamblados
2	2	Biplaza
3	3	Conductor

Ilustración 8. Tabla de la clasificación de los puestos

4.1.4 Puestos

Contiene todos los puestos del taller, con el tiempo en segundos estimado para realizar una unidad de lo que se monte en él. También tiene una columna con el grupo al que pertenece. Algunos no tienen grupo por no considerarse de ninguno de los tres definidos, como por ejemplo "viajes", otros como "limpieza y mantenimiento" y "retrabajos" pueden pertenecer a cualquiera de los grupos, ya que son trabajos que no pertenecen a ningún puesto concreto.

	id_puesto	nombre	tiempo	grupo
1	1	tap_banq_mono		1
2	2	tap_banq_mono_cal		1
3	3	prep_estruct		1
4	4	union_estruct_banq		1
5	5	resp_estruct_lumbar		1
6	6	tap_resp_mono		1
7	7	tap_resp_mono_cal		1
8	8	suspension		1
9	9	guias		1
10	10	tavolino		1
11	11	tap_banq_bip		1
12	12	bip_tav		2
13	13	bip		2
14	14	final_4w_8w_susp		3
15	15	final_8w_susp_cal		3
16	16	final_2w		3
17	17	EXTRA_poli_d2d		2
18	18	carga_prueba		NULL
19	19	limp_mant		NULL
20	20	viajes		NULL
21	21	retrabajos		NULL

Ilustración 9. Tabla de puestos

4.1.5 Puestos físicos

Aquí se recogen las direcciones IP donde se encuentran las atornilladoras de cada puesto, además de la etiqueta que se escanea y registra en la base de datos de las atornilladoras. Esta tabla no está relacionada con ninguna otra, pero será muy útil cuando se desee realizar el recuento de las operaciones realizadas por cada trabajador, ya que, para poder discriminar entre los datos que se registran, se necesitará hacer consultas a los que se guardan en esta tabla. Más adelante se explicará cómo se realiza dicho cómputo con detalle, aquí sólo se muestra la información de la tabla.

	id_puestoFisico	puestoFisico	etiqueta	ip	ip2	ip3	ip4	ip5	ip6
1	1	prep_estruct	RES_ValueIdentifier4	10.73.81.112	NULL	NULL	NULL	NULL	NULL
2	2	union_estruct_banq	RES_ValueIdentifier4	10.73.81.113	NULL	NULL	NULL	NULL	NULL
3	3	suspension	RES_ValueIdentifier3	10.73.81.101	NULL	NULL	NULL	NULL	NULL
4	4	guias	RES_ValueIdentifier3	10.73.81.104	NULL	NULL	NULL	NULL	NULL
5	5	tavolino	RES_ValueIdentifier9	10.73.81.105	NULL	NULL	NULL	NULL	NULL
6	6	bip_tav	RES_ValueIdentifier1	10.73.81.106	10.73.81.107	10.73.81.108	10.73.81.109	NULL	NULL
7	7	bip	RES_ValueIdentifier1	10.73.81.106	10.73.81.107	10.73.81.108	10.73.81.109	NULL	NULL
8	8	final_4w_8w_susp	RES_ValueIdentifier1	10.73.81.102	10.73.81.103	10.73.81.110	10.73.81.111	NULL	NULL
9	9	final_8w_susp_cal	RES_ValueIdentifier1	10.73.81.102	10.73.81.103	10.73.81.110	10.73.81.111	NULL	NULL
10	10	final_2w	RES_ValueIdentifier1	10.73.81.102	10.73.81.103	10.73.81.110	10.73.81.111	NULL	NULL

Ilustración 10. Tabla de puestos físicos

4.1.6 Experiencia

Los operarios en el taller van adquiriendo experiencia en los distintos puestos a medida que pasa el tiempo. Por motivos personales, principalmente lesiones, no todos pueden desempeñarse en todos ellos. Aquí se recoge la experiencia de cada uno en cada puesto, y en caso de que no pueda realizarlo, también se indica.

Las columnas *id_trabajador* e *id_puesto* indican el trabajador y el puesto al que se hace referencia respectivamente. *Disponible* es una columna de tipo booleano, siendo un 1 cuando puede realizarlo y un 0 cuando no. Por último, la columna *experiencia* indica su habilidad en una escala de cero a tres.

	id_experiencia	id_trabajador	id_puesto	disponible	experiencia
16	16	1	16	1	3
17	17	1	17	NULL	NULL
18	18	1	18	0	NULL
19	19	1	19	1	1
20	20	1	20	1	1
21	21	1	21	1	2
22	22	2	1	1	3
23	23	2	2	1	3
24	24	2	3	1	2
25	25	2	4	0	NULL
26	26	2	5	1	3
27	27	2	6	1	2
28	28	2	7	1	2
29	29	2	8	1	3
30	30	2	9	1	3

Ilustración 11. Tabla de experiencia

4.1.7 Índice de las incidencias

Tabla únicamente vinculada con *incidencias*. Se utiliza para guardar los casos más habituales de incidencia. Únicamente recogerá su nombre con una pequeña descripción. Dado que los tiempos son variables e impredecibles, aquí no se indican. Los ejemplos que se muestran en la *Ilustración 12* no son reales.

	id_indice	nombre_incidencia	descripcion
1	1	Médico	Cita médica
2	6	Lesión	Lesión que impide al operario trabajar con norm...
3	7	Picking	El carretillero no está disponible para realizar la ...
4	8	Otros	Incidencia no habitual

Ilustración 12. Tabla del índice de las incidencias

4.1.8 Incidencias

Recoge las incidencias que surjan durante la jornada y los motivos por los que un trabajador se ausente o reduzcan el tiempo que está disponible. La columna *fecha* guardará la fecha en la que suceda, *id_trabajador* e *id_puesto* indican a quién y dónde le sucederá respectivamente, *índice* está referido a la tabla *Índice_incidencias*, *tiempo* es el tiempo que acarrea la incidencia y en *descripción* se puede guardar un comentario. Dependiendo del motivo de la incidencia, el campo que hace referencia al puesto puede estar vacío, dado que si, por ejemplo, tuviera que ir al médico, no se puede encasillar en ningún puesto concreto.

	id_incidencia	fecha	id_trabajador	id_puesto	indice	tiempo	descripcion
1	1	2021-05-18	29	NULL	7	2700	El carretillero no está disponible
2	2	2021-05-25	14	13	6	24000	Lesión muscular
3	3	2021-07-03	5	18	8	1800	Mal funcionamiento de la máquina
4	4	2021-07-04	12	NULL	1	5400	Motivos personales
5	5	2021-07-10	3	14	6	1800	Leve golpe
6	6	2021-07-14	9	NULL	1	3600	Vacuna contra el COVID-19
7	7	2021-07-16	22	NULL	7	2700	El carretillero no está disponible

Ilustración 13. Tabla de incidencias

4.1.9 Referencias de los asientos

Recoge todas las referencias de los asientos que se fabrican. Con esta tabla sucede algo similar a lo que ocurre en la de puestos físicos, ya que estos datos únicamente se recogen para realizar consultas cuando se vayan a hacer recuentos de los asientos.

	id_asiento	nombre	tipo	puesto	id_2019	id_2021
11	11	S2019 A.N. 1P.8W SX TX D2D Borgo Agnello	ENS. FINAL 4W-8W-8WSUS	14	5802802564	5802825838
12	12	S2019 A.N. 1P.8W DX TX D2D Borgo Agnello	ENS. FINAL 4W-8W-8WSUS	14	5802802565	5802825965
13	13	S2019 A.N. 1P.8W SX FP	ENS. FINAL 4W-8W-8WSUS	14	5802802567	5802825977
14	14	S2019 A.N. 1P.8W DX FP	ENS. FINAL 4W-8W-8WSUS	14	5802802568	5802825984
15	15	S2019 A.N. 1P.8W SUSP.SX FP	ENS. FINAL 4W-8W-8WSUS	14	5802802630	5802826181
16	16	S2019 A.N. 1P.8W SUSP.DX FP	ENS. FINAL 4W-8W-8WSUS	14	5802802631	5802826185
17	17	S2019 1P.2W SX GDX TX APT PUR	ENS. FINAL 2W	16	5802238656	NULL
18	18	S2019 1P.2W DX GSX TX APT PUR	ENS. FINAL 2W	16	5802238657	NULL
19	19	S2019 A.N. 1P.8W SUSP. SX TX H APT PUR	ENS. FINAL 8WSUS CALEF	15	5802802623	5802826141
20	20	S2019 A.N. 1P. 8W SUSP SX TX H APT IN SI...	ENS. FINAL 8WSUS CALEF	15	5802802628	5802826148
21	21	S2019 A.N. 1P.8W SUSP. DX TX H APT PUR	ENS. FINAL 8WSUS CALEF	15	5802802625	5802826158
22	22	S2019 A.N. 1P.8W SUSP.DX RIS APT IN SITU	ENS. FINAL 8WSUS CALEF	15	5802802629	5802826174
23	23	S2019 A.N. 2P. DX SX BIS PRET. TX NO TA...	BIPO BIS	13	5802813800	5802825438
24	24	S2019 A.N. 2P. SX GDX BIS PRET. TX NO T...	BIPO BIS	13	5802813806	5802825440
25	25	S2019 A.N. 2P. DX GSX BIS S/PRET TX NO ...	BIPO BIS	13	5802813807	5802825447
26	26	S2019 A.N. 2P. SX GDX BIS S/PRET TX NO ...	BIPO BIS	13	5802813808	5802825453
27	27	S2019 A.N. 2P.DX BIS PRET. NO TAV TX D2D	BIPO BIS	13	5802826987	5802826988
28	28	S2019 A.N. 2P.SX BIS PRET NO TAV TX D2D	BIPO BIS	13	5802825468	5802825443
29	29	S2019 A.N. 2P.DX BIS S/PRET NO TAV TX D...	BIPO BIS	13	5802825476	NULL
30	30	S2019 A.N. 2P.DX GSX BIS PRET. TX APT P...	BIPO BIS TAVOLINO	12	5802813809	5802825456
31	31	S2019 A.N. 2P.DX GSX BIS PRET. TX APT I...	BIPO BIS TAVOLINO	12	5802813813	5802825458
32	32	S2019 A.N. 2P.SX BIS PRET. TX	BIPO BIS TAVOLINO	12	5802813815	5802825498
33	33	S2019 A.N. 2P.DX GSX BIS PRET. TX APT I...	BIPO BIS TAVOLINO	12	5802813816	5802825506

Ilustración 14. Tabla de referencias de los asientos

4.1.10 Planificación

Esta es una de las novedades que se introduce en la empresa. Actualmente el encargado decide el trabajo que va a realizar cada uno en la jornada y en qué puestos, guardándolo en un Excel. La novedad es que pasará a guardarlos desde la página web interna que se crea en este proyecto y se almacenarán en esta tabla. Estos datos son *fecha*, *id_trabajador*, *id_puesto* y *número,* haciendo referencia a la fecha en la se realiza el trabajo, el trabajador, puesto y unidades que se montan de lo correspondiente a dicho puesto. Los operarios nunca realizan una jornada completa en el mismo puesto, por lo que habrá varias filas haciendo referencia a un mismo trabajador, pero en distintos puestos.

	id_orden	fecha	id_trabajador	id_puesto	numero
893	4381	2021-07-27	17	3	45
894	4382	2021-07-27	23	3	45
895	4383	2021-07-27	17	2	50
896	4384	2021-07-27	2	1	50
897	4385	2021-07-27	2	2	25
898	4386	2021-07-27	2	3	30
899	4387	2021-07-27	15	1	24
900	4388	2021-07-27	15	2	50
901	4389	2021-07-27	15	4	42
902	4390	2021-07-26	14	20	54

Ilustración 15. Tabla de planificación

4.1.11 Registros

Esta es otra de las principales novedades. Permitirá acceder tanto a datos antiguos como actuales y comprobar cualquiera de ellos, pudiendo hacerse un seguimiento en tiempo real de la producción de cada trabajador. Sus columnas son:

- **Fecha**: para indicar el día al que se hace referencia.
- **Id_trabajador**: operario que desempeña el trabajo.
- **Id_puesto**: puesto en el que se desempeña.
- **Cantidad_total**: unidades de producto que debe realizar en la jornada en dicho puesto. Este dato se obtendrá de la tabla *planificación* en su columna *número*.
- **Cantidad_realizada**: unidades que en ese momento ya ha realizado.
- **Tiempo_total**: cada unidad de producto tiene un tiempo estimado (columna *tiempo* en la tabla *puestos*). Las unidades que debe montar en ese puesto (*cantidad_total*) en la jornada completa multiplicadas por el tiempo estimado para cada una generan *tiempo_total*.

$$tiempo\_total = cantidad\_total \cdot tiempo$$

- **Tiempo_empleado**: se calcula igual que *tiempo_total*, con la diferencia de que en este caso se calcula con las unidades que ya se han realizado (*cantidad_realizada*) en lugar de con las totales.

$$tiempo\_empleado = cantidad\_realizada \cdot tiempo$$

- **Tiempo_restante:** se trata del tiempo estimado que falta para terminar de producir lo planificado en ese puesto. Es la diferencia entre el *tiempo_total* y el *tiempo_empleado*.

$$tiempo\_restante = tiempo\_total - tiempo\_empleado$$

- **Producción**: porcentaje producido en ese instante del total que se tiene que producir en la jornada en ese puesto.

$$producción = \frac{cantidad\_realizada}{cantidad\_total} \cdot 100$$

- **Productividad**: valor en tanto por ciento de la carga de trabajo del puesto sobre el tiempo disponible del trabajador. El tiempo disponible está almacenado en la tabla de "registros conjuntos" en el capítulo *4.1.12*.

$$productividad = \frac{tiempo\_total}{tiempo\_disponible} \cdot 100$$

	id_registro	fecha	id_trabajador	id_puesto	cantidad_total	cantidad_realizada	tiempo_total	tiempo_empleado	produccion	productividad	tiempo_restante
1	2636	2021-07-30	16	14	15	13	13500	11700	86.6667	50.0000	1800
2	2637	2021-07-30	16	15	8	2	9120	2280	25.0000	33.7778	6840
3	2638	2021-07-30	16	3	20	0	4800	0	0.0000	17.7778	4800
4	2639	2021-07-30	50	8	50	38	15600	11856	76.0000	57.7778	3744
5	2640	2021-07-30	50	9	64	0	11136	0	0.0000	41.2444	11136
6	2641	2021-07-30	55	10	40	40	13920	13920	100.0000	51.5555	0
7	2642	2021-07-30	55	3	25	9	6150	2214	36.0000	22.7778	3936
8	2643	2021-07-30	55	6	10	0	2280	0	0.0000	8.4444	2280
9	2644	2021-07-30	55	1	20	0	4680	0	0.0000	17.3333	4680
10	2645	2021-07-30	19	14	10	6	9000	5400	60.0000	40.0000	3600
11	2646	2021-07-30	19	15	7	3	7980	3420	42.8571	35.4667	4560
12	2647	2021-07-30	19	5	44	44	5544	5544	100.0000	24.6400	0
13	2648	2021-07-30	25	13	14	9	13524	8694	64.2857	50.0888	2898
14	2649	2021-07-30	25	12	9	6	11610	7740	66.6667	43.0000	3870
15	2650	2021-07-30	25	4	12	0	2160	0	0.0000	8.0000	2160
16	2651	2021-07-30	21	18	33	17	13266	6834	51.5151	49.1333	6432
17	2652	2021-07-30	21	11	55	35	13200	8400	63.6363	48.8888	4800

Ilustración 16. Tabla de registros

4.1.12 Registros conjuntos

En la tabla de registros ciertos datos no son todo lo útiles que podrían ser. Esto es debido a que para poder mostrar la cantidad realizada en cada puesto, cada uno de ellos ha de ser una fila distinta, y ciertos datos son mucho más prácticos si se muestran de forma conjunta, haciendo referencia a la jornada completa y no a cada puesto. Por todo ello en esta tabla no se hace referencia a ninguno concreto, sólo al trabajador, y se eliminan las columnas que muestran cantidades, dado que, por la distinta complicidad de cada uno, no es un dato que se pueda comparar de forma global.

- **Fecha**: para indicar el día al que se hace referencia.
- **Id_trabajador**: operario que desempeña el trabajo.
- **Tiempo_total**: sumatorio de tiempos totales en cada puesto de cada trabajador.

$$tiempo\_total_{Registros\ conjuntos} = \sum tiempo\_total_{Registros}$$

- **Tiempo_empleado**: sumatorio de tiempos empleados en cada puesto de cada trabajador.

$$tiempo\_empleado_{Registros\ conjuntos} = \sum tiempo\_empleado_{Registros}$$

- **Tiempo_disponible**: tiempo en minutos que el operario va a trabajar esa jornada. Por defecto son 450 minutos, es decir, ocho horas descontando los descansos.
- **Tiempo_restante:** se trata del tiempo restante disponible. Es la diferencia entre el tiempo_disponible y el tiempo_empleado, es decir el tiempo estimado que le queda al trabajador para terminar la producción exigida.

$$tiempo\_restante = tiempo\_disponible - tiempo\_empleado_{Registros\ conjuntos}$$

- **Producción**: porcentaje producido en ese instante respecto al tiempo disponible en la jornada.

$$producción = \frac{tiempo\_empleado_{Registros\ conjuntos}}{tiempo\_disponible} \cdot 100$$

- **Productividad**: dato en el que se basa el encargado cuando determina la cantidad de trabajo que tiene que realizar cada empleado. Asigna el número de unidades que se deben producir en cada puesto buscando que este dato quede lo más cercano que sea posible al 100%. Es muy difícil que consiga asignar exactamente el 100%, dado que los tiempos que generan *tiempo_total* está preasignados en cada puesto. Este mismo dato también se muestra en la tabla "planificación", en el capítulo *4.1.10*.

$$productividad = \frac{tiempo\_total_{Registros\ conjuntos}}{tiempo\_disponible} \cdot 100$$

- **Ritmo**: el objetivo con esto es comprobar en qué momento de la producción se encuentra el trabajador si nos referimos a lo que ha producido y lo comparamos con lo que debería llevar respecto al tiempo. Por ejemplo, si han pasado cuatro horas de las ocho que hace la jornada completa, debería llevar la mitad de la producción, siendo ésta el 100%. Si no llega a la mitad estará por debajo del 100% mientras que si lleva más estará por encima. En contra de lo que pueda parecer, el objetivo es que el operario esté lo más cerca posible del 100%, dado que lo habitual es que estén la mayoría de la jornada por encima y les sobre tiempo antes de acabar el turno, desencadenando posibles lesiones.

  En la fórmula a continuación, el tiempo desde el inicio es la variable que representa el tiempo que ha pasado desde el comienzo del turno hasta el instante en el que se está calculando el dato.

$$ritmo = \frac{\sum tiempo\_empleado}{Tiempo\ desde\ el\ inicio} * 100$$

	id_registro	fecha	id_trabajador	tiempo_total	tiempo_empleado	tiempo_disponible	produccion	productividad	ritmo	tiempo_restante
1	1356	2021-07-30 ...	16	27420	13980	27000	51.7778	101.5556	97.9	13020
2	1357	2021-07-30 ...	50	26736	11856	27000	43.9111	99.0222	83.0	15144
3	1358	2021-07-30 ...	55	27030	16134	27000	59.7556	100.1111	113.0	10866
4	1359	2021-07-30 ...	19	22524	14364	22500	63.8400	100.1066	100.6	12636
5	1360	2021-07-30 ...	25	27294	16434	27000	60.8666	101.0889	115.1	10566
6	1361	2021-07-30 ...	21	26466	15234	27000	56.4222	98.0222	106.7	11766

Ilustración 17. Tabla de registros conjuntos

4.2 Creación de la *REST-API*

Desde este punto se trabajará principalmente con Visual Studio Code, y con él desarrollaremos la *REST-API.*

4.2.1 Acceso a la base de datos

Lo primero que debe hacerse es conectar la *REST-API* a Microsoft Server SQL, ya que es el programa que se ocupa de la gestión de la base de datos. Para ello se crea un archivo *database.ts* donde se alojará el código con la información necesaria para realizar la conexión con el programa. Esto es el nombre del servidor en el que se encuentra la base de datos, usuario y contraseña de la instancia y la dicha base de datos a la que nos conectaremos. En la *Ilustración 18* se ve el código necesario para realizar dicha conexión.

```
3  const pool = require('mssql');
4
5  try {
6      pool.connect('mssql://    :    @                /          /                ?encrypt=true');
7      console.log('Se ha conectado a la base de datos');
8  } catch (error) {
9      console.log(error)
10 }
11 export default pool;
```

Ilustración 18. Código de conexión a la base de datos

4.2.2 Creación de los controladores

Su función es la de interactuar directamente con MSSQL, al que envía código para así manipular la información. Éste será para leer las tablas (enviaremos solicitudes *SELECT*), para introducir nuevos datos (*INSERT INTO*), para actualizar los ya existentes (*UPDATE*) y para eliminarlos (*DELETE FROM*). La complejidad de las órdenes enviadas variará en función de lo que se quiera obtener de vuelta. Por ejemplo, en una instrucción de lectura es más sencillo pedir que simplemente mande toda la información de una tabla que seleccionar algunos datos y cruzarlos con los de otras. Estas órdenes se denominan métodos, las cuales se crean dentro de una clase, que a su vez se almacena en una constante que se exporta.

En la *Ilustración 19* se muestra como ejemplo un método de tipo *GET* de la tabla de encargados, siendo la zona anaranjada el código en lenguaje SQL que se envía a la base de datos.

```
public async list (req: Request, res: Response) {
    const lectura = await pool.query(`
    USE Datos_valladolid;
    SELECT * FROM Encargados`);
    console.log(lectura.recordset);
    res.json(lectura.recordset);
}
```

Ilustración 19. Ejemplo de método del controller

Estos controladores se crearán para todas y cada una de las tablas existentes en la base de datos, ya que, si no, nuestro programa no podría interactuar con ellas. En la *Ilustración 20* se pueden ver todos los controladores que se han creado.

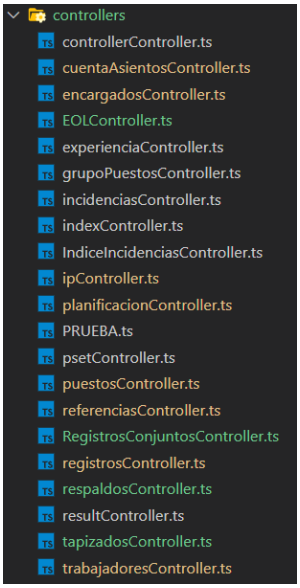


Ilustración 20. Archivos de controladores

4.2.3 Creación de las rutas

En un archivo índice (*index.ts*) se recogen todas las rutas generales de cada tabla, desde las cuales se hace una redirección a las rutas más concretas de cada tabla.

En los archivos de ruta (*routes.ts*) se genera una ruta para cada método creado antes. Como ya se explicó con anterioridad, las órdenes que se dan pueden ser de cuatro tipos: *GET, POST, PUT* y *DELETE.* De esta forma, cuando el método al que se hace referencia es de lectura se utilizará *GET*, cuando sea para insertar datos se utiliza *POST*, si es para actualizar datos es *PUT* y si es para eliminarlos *DELETE*. En el método de inserción de datos no es necesario dar información adicional, ni en el de lectura si se quiere obtener toda la información de la tabla. No obstante, para eliminar o editar algún dato sí que hace falta decir a cuál en concreto se hace referencia, indicándose en la ruta. Aquí se muestra la importancia de crear las claves principales o *primary keys* que se explicó con anterioridad, ya que se indica el dato que queremos manipular a través de ellas.

En definitiva, se indica el tipo de orden, la ruta, la constante importada del controlador y el método al que se refiere. Igual que en anteriormente con los controladores, las rutas se crean métodos dentro de clases, que también se guardan en una constante que se exporta. En la *Ilustración 21* se muestra como ejemplo el método de las rutas creadas para la tabla de encargados. En la *Ilustración 22* se ven todos los archivos dedicados a las rutas.

```
config(): void {
    this.router.get('/', encargadosController.list);
    this.router.get('/:id', encargadosController.getOne);
    this.router.post('/', encargadosController.create);
    this.router.put('/:id', encargadosController.update);
    this.router.delete('/:id', encargadosController.delete);
}
```

Ilustración 21. Rutas de la tabla de encargados

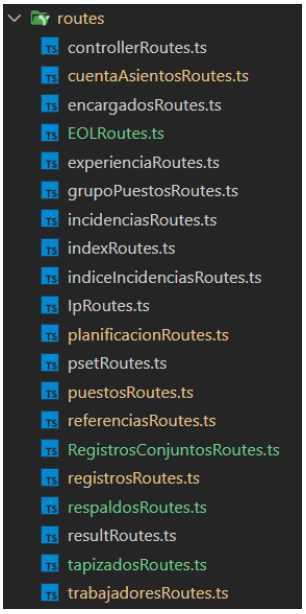


Ilustración 22. Archivos de rutas

La constante exportada llega a un archivo denominado *index.ts*, donde se indican rutas más concretas para cada tabla (*Ilustración 23*).

```
routes(): void {
    this.app.use('/', indexRoutes);
    this.app.use('/api/encargados', encargadosRoutes);
    this.app.use('/api/trabajadores', trabajadoresRoutes);
    this.app.use('/api/grupoPuestos', grupoPuestosRoutes);
    this.app.use('/api/puestos', puestosRoutes);
    this.app.use('/api/experiencia', experienciaRoutes);
    this.app.use('/api/registros', registrosRoutes);
    this.app.use('/api/incidencias', incidenciasRoutes);
    this.app.use('/api/indiceincidencias', indiceIncidenciasRoutes);
    this.app.use('/api/planificacion', planificacionRoutes);
}
```

Ilustración 23. Ruta de cada tabla

En definitiva, existe una ruta para cada tabla en el archivo *index.ts*, que dirige a las de cada una. Estas conectarán con los controladores, que enviarán el código a MSSQL para que realice la orden deseada.

4.2.4 Comprobación del funcionamiento

Para ello utilizaremos el programa Postman, cuya finalidad es precisamente esta, comprobar el funcionamiento de las *REST-API*. En la *Ilustración 24* se ve la pantalla inicial de Postman.

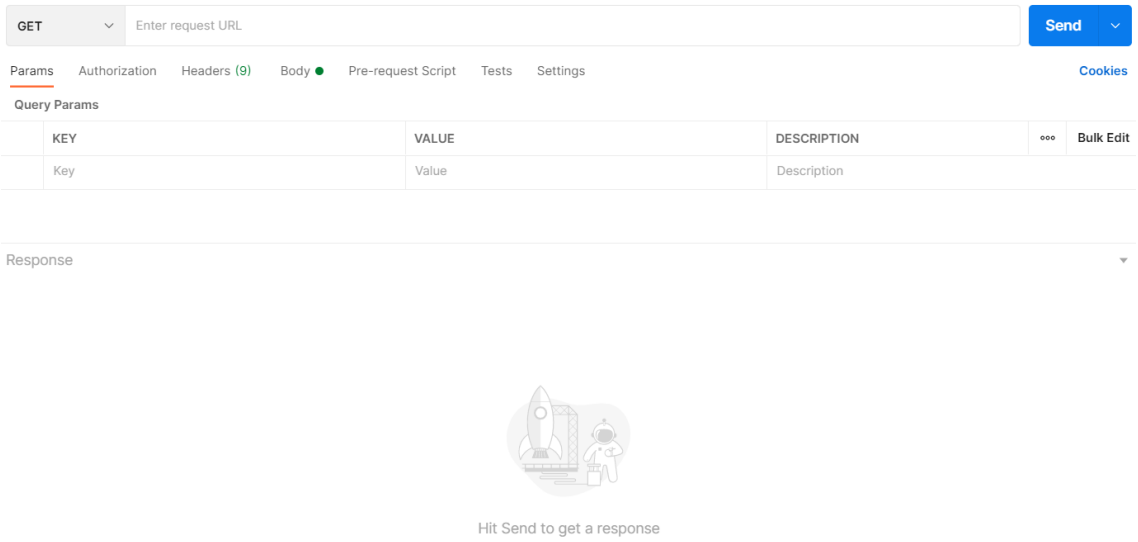


Ilustración 24. Introducción de Postman

Su funcionamiento es sencillo. En primer lugar, deberá seleccionarse el tipo de orden que se desea dar, es decir, de tipo *GET*, *POST*, *PUT* y *DELETE* en nuestro caso.

A continuación, debe introducirse el servidor y puerto en el que nos encontramos, seguido de las consabidas rutas que hemos creado. Dado que se está trabajando en un entorno de prueba en un servidor local, éste será *localhost*. El puerto está configurado en el archivo *index.ts* del que se habló con anterioridad, siendo éste el puerto 3000. En el caso de que se desee simplemente hacer una lectura global, esto sería suficiente. Como vemos en *Ilustración 25*, lo que el programa nos devuelve es la información de tipo *.json* que estamos creando con los controladores anteriores.

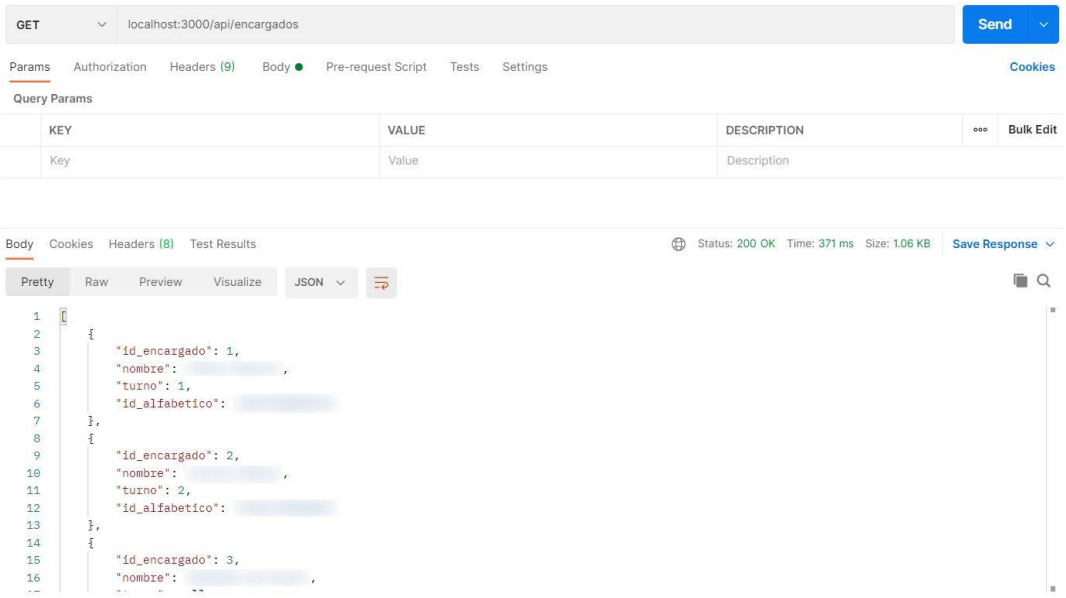


Ilustración 25. Ejecución GET en Postman

Si quisiera introducirse un nuevo dato a la tabla, se escribe manualmente el *.json*, imitando lo que recibiría si estuviera recibiendo datos desde la web. Postman devuelve el .json del dato enviado por estar así configurado en el controlador, como podemos ver en la *Ilustración 26*.

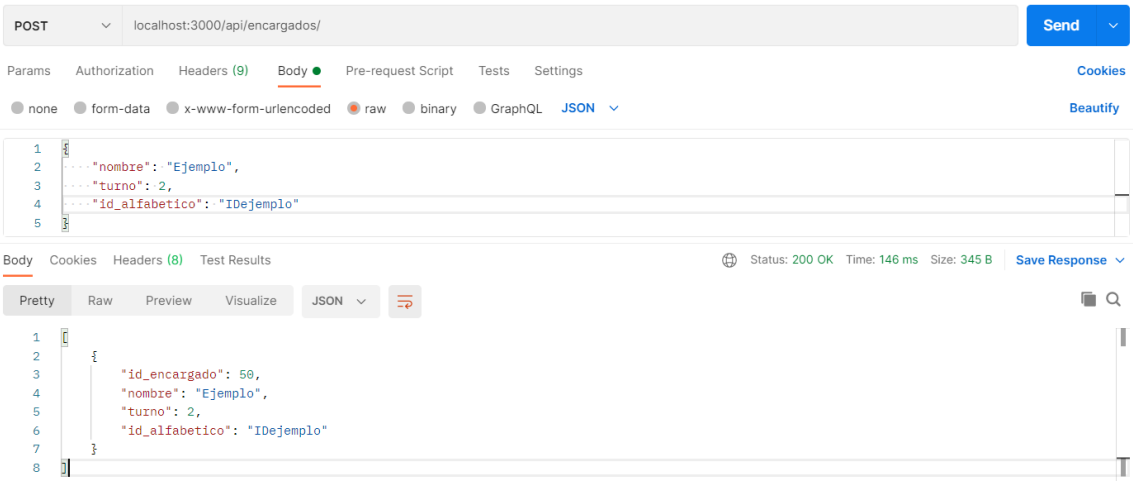


Ilustración 26. Ejecución POST en Postman

En caso de que se quisiera actualizar un dato, debe añadirse el .json con los datos actualizados. Para que el programa sepa qué dato debe cambiar, se añade a la ruta el ID, es decir, la clave principal de la fila generada automáticamente. Todo ello se ve en la *Ilustración 27*.

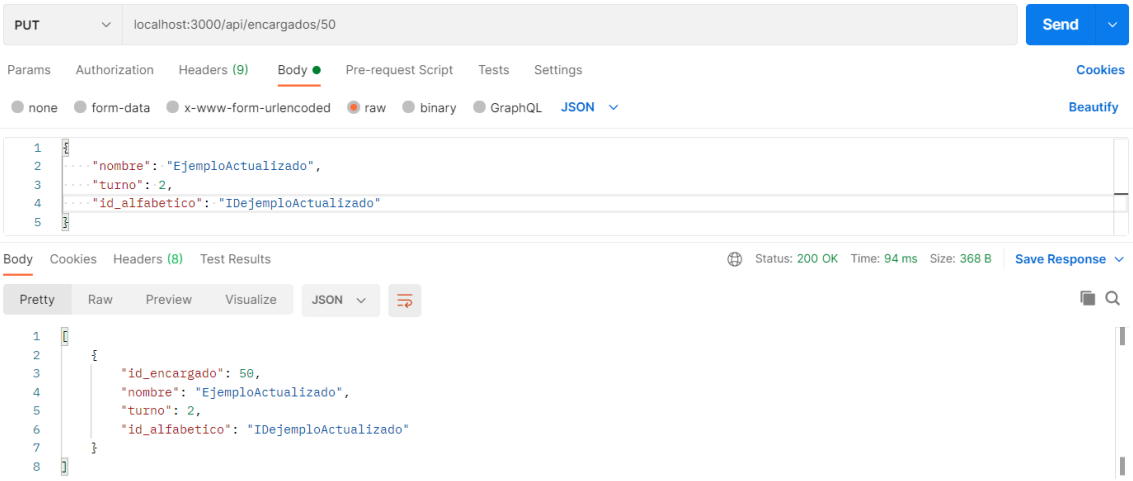


Ilustración 27. Ejecución PUT en Postman

Si deseara eliminarse el dato también sería necesario añadir el ID a la ruta. El texto que devuelve también ha sido programado en el controlador. Visible en la *Ilustración 28*.

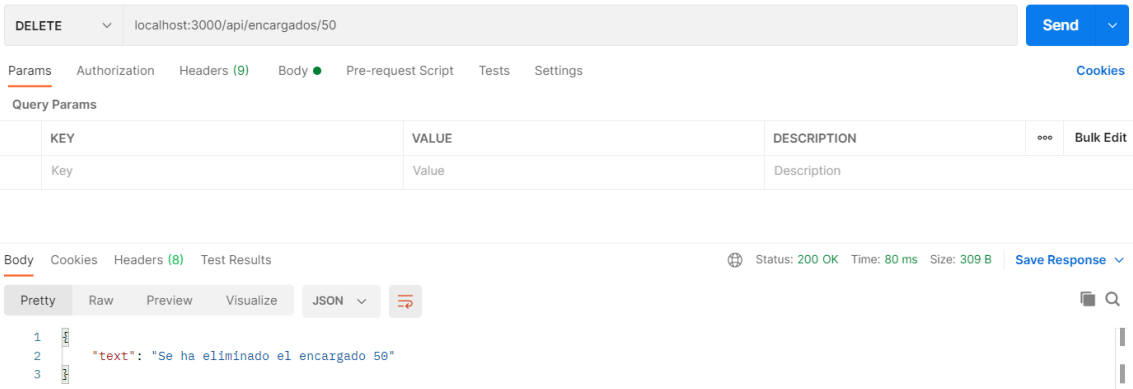


Ilustración 28. Ejecución DELETE en Postman

4.3 Nociones básicas del funcionamiento

En este punto comenzamos a desarrollar el *frontend*. Ya está creada la base de datos y el servidor, por lo que lo siguiente es crear la interfaz para poder utilizarla. Esto se hará con Angular. Hay que destacar que esta parte del proyecto puede considerarse hasta cierto punto independiente de la anterior, pudiendo incluso ser desarrolladas por equipos diferentes.

4.3.1 Angular Material

Angular es una plataforma de desarrollo construida sobre TypeScript. Sus partes principales se desarrollan a continuación.

4.3.1.1 Los componentes

Cada uno de ellos contiene cuatro archivos de código en su interior, y su combinación da lugar a una ventana nueva en la aplicación. Se crean automáticamente desde la consola con el comando "*ng generate components*" seguido del nombre que se desee dar a los archivos. El conjunto de los cuatro se muestra en la *Ilustración 29* y son:

- **HTML:** proveniente del inglés "*HyperText Markup Language"* o *"Lenguaje de Marcado de Hipertexto"* en español, es el lenguaje para el desarrollo de páginas de internet. Con él se desarrollará la estructura de la página, para que posteriormente el navegador lo interprete y dé forma a la pantalla. En él se introducirá todo lo que se muestre por pantalla, es decir, los textos, tablas, botones, etc.

- **CSS:** del inglés "C*ascading Style Shits"*, u "Hojas de Estilo en Cascada", sirve para dar formato o diseño a todo lo establecido en el HTML. Esto constituye el color, tamaño, fuente, posición, etc.

- **SPEC:** se trata de un archivo de prueba de la unidad y se utiliza cuando se ejecuta el comando correspondiente (*ng test*). En este proyecto no se utilizará.

- **TS:** archivo de TypeScript. Se trata del principal del componente, en él se desarrollan todas las funciones que están detrás del componente.

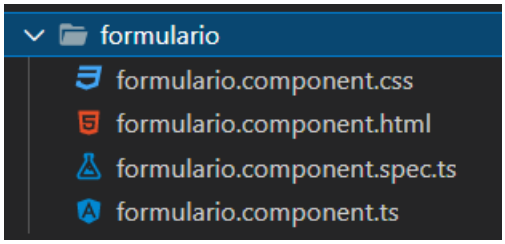


Ilustración 29. Ejemplo de componente

4.3.1.2 Las plantillas

Muy utilizadas en el proyecto. De ellas se obtendrán todos los elementos dinámicos de la página, tales como las tablas de la base de datos, hasta la barra lateral de navegación, los

elementos de autocompletado y un largo etc. En la página web se puede ver una lista de todas las plantillas que incluye.

Una vez que se selecciona la que se desea utilizar, se encuentran múltiples ejemplos de posibles formas de utilizarlas, de más simples a más complejas y variadas. Se puede ver el código que se esconde detrás de cada uno de los ejemplos, que además viene desglosado según lo que necesite cada uno de los archivos del componente.

4.3.1.3 Los inyectables

Permiten declarar las dependencias de las clases creadas sin necesidad de instanciarlas, dado que ya lo hace Angular. Esto se utilizará en las clases de los servicios, cuyo funcionamiento se explicará más adelante.

4.3.1.4 Angular CLI

Comandos que se introducen en la consola y son de altísima utilidad. Todos ellos se utilizarán únicamente en el *frontend*, ya que es donde se utiliza Angular. Únicamente se van a explicar los comandos que se utilizan en este proyecto:

- ***ng build*:** compila la aplicación a un directorio de salida. En este caso se utiliza cuando se da el paso de cambiar la aplicación del ordenador en el que se trabaja en local a subirlo al servidor, generando los archivos necesarios para poder desplegarla.

- ***ng serve*:** fundamental en el desarrollo del proyecto. Construye la aplicación y la despliega como si estuviera en un servidor, además de reconstruirla cuando se hacen cambios. Se puede acceder y navegar en ella a través de la dirección indicada. En este caso, dicha dirección es por defecto "*http://localhost:4200/*". De esta forma se puede ir viendo y comprobando todo lo que se trabaja nada más realizarlo.

- ***ng generate*:** genera o modifica archivos basados en un esquema. Se utilizará para crear todos los componentes, explicados con anterioridad, y los servicios, explicados a continuación. Los componentes se generan con el comando "*ng generate components*" seguido del nombre que se desee dar a los archivos, mientras que los servicios se crean con "*ng generate services*", también seguido del nombre del servicio.

4.3.2 Los servicios

Un servicio es "*una amplia categoría que abarca cualquier valor, función o característica que necesite una aplicación*"(Angular, s. f.). Normalmente es una clase con propósito estrecho y bien definido. Debe hacer algo específico y hacerlo bien.

En este proyecto se va a usar para obtener datos del servidor, es decir, como unión entre el *backend* y *frontend*. Previamente se explicó que los métodos creados en los *controller* son llamados a través rutas para poder utilizarlos. En estos archivos se crean funciones en las que se introducen dichas rutas, de forma que cuando se necesita interactuar con la base de datos desde el archivo de tipo .ts, se llama a las funciones aquí creadas, que realizarán la acción que corresponda.

Una de las ventajas es que, al ser las tareas definidas en una clase de servicio inyectable, pueden ser utilizadas desde cualquier componente.

Se crean tantos servicios como *controllers.* Podrían introducirse todos en el mismo, pero para hacer el código más legible se recomienda que se dividan.

A continuación se va a mostrar un ejemplo de la cadena de llamadas que se realiza cuando quiere obtenerse la información de la base de datos. En este caso es la de obtener un trabajador a través de su id, pero cualquier consulta se realizaría de la misma forma:

Este primer fragmento de código, mostrado en la *Ilustración 30,* se encontraría en la componente de tipo .ts. En él se hace una llamada al servicio "*getTrabajador"*, enviando la variable entre paréntesis "*id_trabajador*". La información que se obtiene de vuelta llega en la variable "*res"* (*response,* respuesta), pudiendo utilizarla como se desee. En este caso, al ser un ejemplo, simplemente se mostrará en la consola.

```
this.trabajadoresService.getTrabajador(id_trabajador).subscribe(
  res => {
    console.log(res)
  }
)
```

Ilustración 30. Llamada al servicio getTrabajador

Tras la llamada llegaríamos al propio servicio, *Ilustración 31*, que simplemente realizaría otra a la ruta predefinida con anterioridad y que corresponde a la consulta que se quiera hacer.

```
getTrabajador(id: string){
  return this.http.get(`http://localhost:3000/trabajadores/${id}`);
}
```

Ilustración 31. Servicio llamando a una ruta

La *Ilustración 32* pertenece al *backend*. En ella se muestra el listado de rutas a los correspondientes *controllers*, marcando con una flecha la que se ha indicado desde el servicio. Como podemos ver, hace referencia a la función "*getOne*", que se mostrará a continuación.

```
config(): void {
    this.router.get('/', trabajadoresController.list);
    this.router.get('/:id', trabajadoresController.getOne);
    this.router.get('/nombre/:nombre', trabajadoresController.getId);
    this.router.get('/turno/:id', trabajadoresController.turno);
    this.router.post('/', trabajadoresController.create);
    this.router.put('/:id', trabajadoresController.update);
    this.router.delete('/:id', trabajadoresController.delete);
}
```

Ilustración 32. Listado de rutas de trabajadores

En el inicio de la *Ilustración 33* se ve el nombre de la función "getOne", y en la parte central, la instrucción en lenguaje SQL que se envía a la base de datos. El dato enviado desde el *frontend* se recoge en la variable "*id*", que se inserta en el código SQL. De esta forma la base de datos devolvería el dato solicitado en formato .json, que llegaría a la variable "*res*" de la componente inicial, tal y como se explicó con anterioridad.

```
public async getOne (req: Request, res: Response) {
    const { id } = req.params;
    const lectura = await pool.query(`
    USE Datos_valladolid;

    SELECT * FROM Trabajadores
    WHERE id_trabajador = '${id}'`);
    console.log(lectura.recordset);
    res.json(lectura.recordset);
}
```

Ilustración 33. Función del controller de trabajadores

4.4 La aplicación

A continuación se va a mostrar con detalle cada página de la aplicación, los elementos de cada página y cómo funciona.

4.4.1 Inicio

Ilustración 34. Página de inicio

La *Ilustración 34* es la página que se ve nada más abrir la aplicación. Contiene el encabezado o *navBar* de Bootstrap y la barra de navegación lateral o *side-nav* de Angular Material.

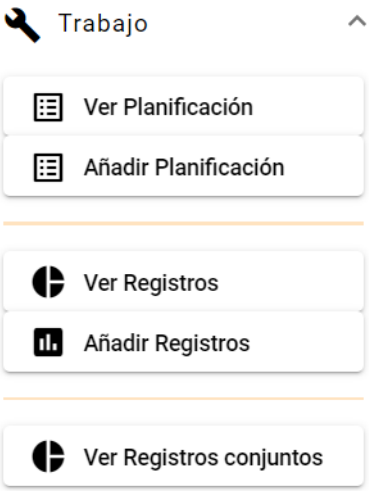
El encabezado contiene la palabra ISRI a la izquierda, que a su vez hace de botón que redirecciona siempre a esta página, y Valladolid a la derecha, como simple elemento decorativo.

La barra de navegación lateral, que se expande y se contrae con los botones que se encuentran en la zona superior, es fundamental para poder cambiar de página. Existe una pestaña para cada tabla de la base de datos, excepto para la de "Clasificación de puestos" y "Puestos físicos". Esto se debe a que ambas tablas son fijas, y además, modificar "Puestos físicos" tiene consecuencias directas en el funcionamiento de la aplicación, por lo que conviene que su modificación no sea excesivamente accesible.

A continuación, en la *Ilustración 35*, se muestra la barra desglosada:

Navegación lateral: Personal

Navegación lateral: Reparto



Navegación lateral: Trabajo

Ilustración 35. Navegación lateral

Las ventanas "Personal" (*Navegación lateral: Personal*) y "Reparto" (*Navegación lateral: Reparto*) son iguales en cuanto a programación y creación se refiere, siendo su contenido el único motivo para dividirlas. Cada tabla tiene dos pestañas, una para ver su contenido y otra para crearlo.

4.4.2 Elementos de la visualización de los datos

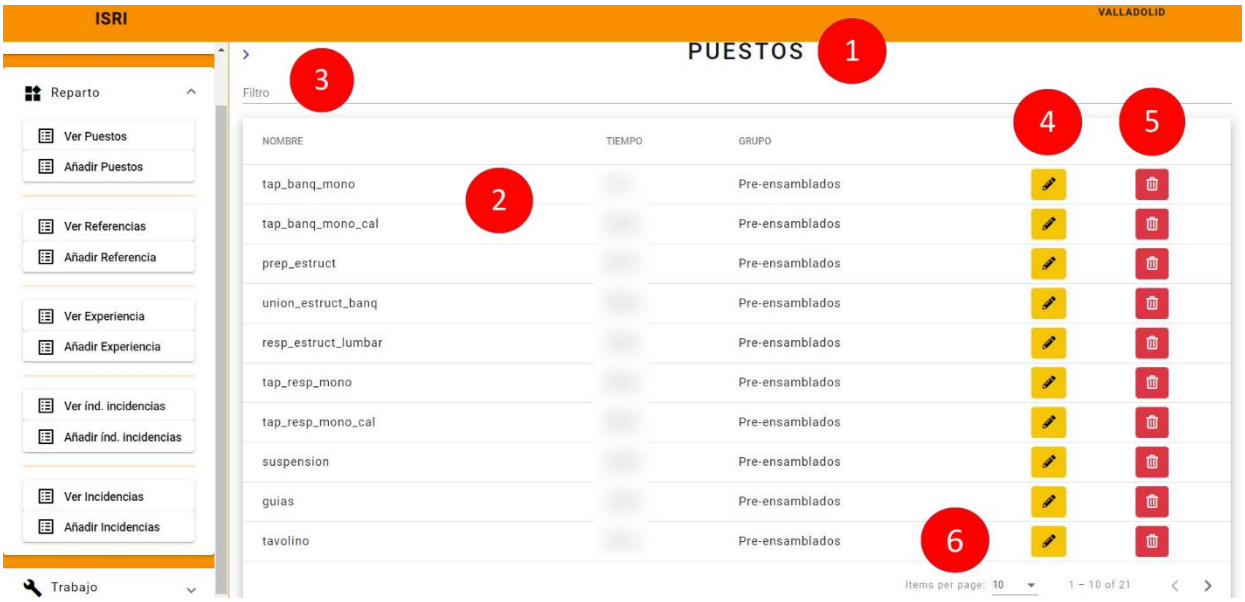


Ilustración 36. Explicación de la visualización de los datos

En la *Ilustración 36* se encuentra el conjunto de visualización de los datos con los elementos numerados, que se explican a continuación:

1- **Título:** nombre de la tabla que se representa a continuación.
2- **Tabla:** en ella se ve el contenido de la tabla de la base de datos, pudiendo no ser idéntica debido a posibles cambios en la consulta. La columna del id de cada fila no se muestra, ya que es importante pero solamente de cara a la programación interna. El resto de posibles cambios se explicarán más adelante.
3- **Filtro:** permite filtrar por cualquier valor de cualquier columna de la tabla. En *Ilustración 37* se muestra un ejemplo de filtrado por la columna "grupo".

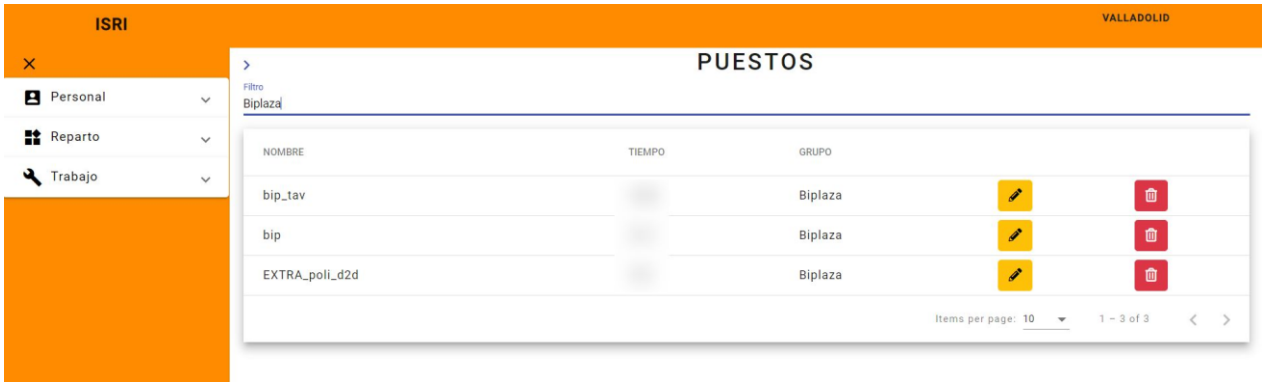


Ilustración 37. Ejemplo de filtrado

4- **Botón de editar:** redirige al formulario con el que se insertan los datos, con la diferencia de que en este caso los huecos ya están rellenos. Al tocar el botón "Guardar", en lugar de crear una nueva fila en la tabla, modifica la que se ha seleccionado. Posteriormente redirige a la tabla automáticamente.
5- **Botón de eliminar:** elimina la fila seleccionada.

6- **Páginas:** tiene dos botones. El primero permite seleccionar el número de filas que se muestran por página, por defecto 10, mientras que el otro permite cambiar de página. Se ve en la *Ilustración 38*.

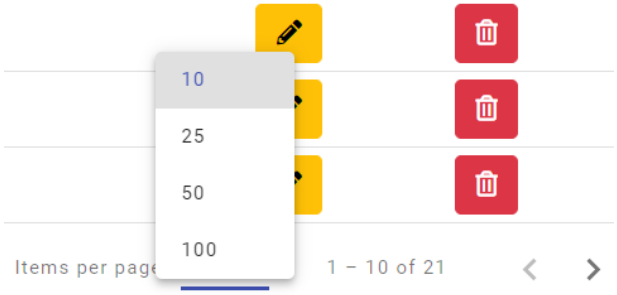


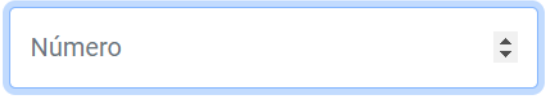
Ilustración 38. Páginas y elementos en ellas

4.4.3 Elementos para añadir o editar contenido

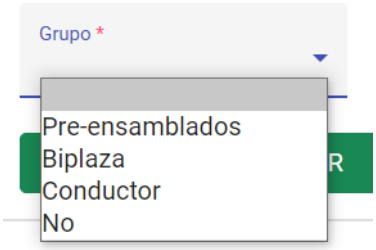
Estas ventanas varían en función de la tabla seleccionada y los datos que se vayan a añadir o editar. En la *Ilustración 39* se encuentra como ejemplo el formulario con el que se añadiría una incidencia. Igual que en las de visualización de datos, en la parte superior está el título que indica ventana en la que se está operando y en la inferior el formulario que se ha de rellenar para enviar la información a la base de datos. La ventana es la misma si se quiere añadir un dato o editarlo, la diferencia se establece al acceder a ella. Si se hace desde el menú, ésta estará vacía y creará un nuevo registro, mientras que si se hace desde uno de los botones de editar de cualquiera de las tablas, ésta aparecerá previamente rellena con los datos que se van a editar. Al darle a guardar, en el primer caso se ejecuta un *POST*, mientras que en el segundo se ejecuta un *PUT*. Existen diferentes tipos de campos según el tipo de dato:

Ilustración 39. Explicación del formulario para añadir o editar contenido

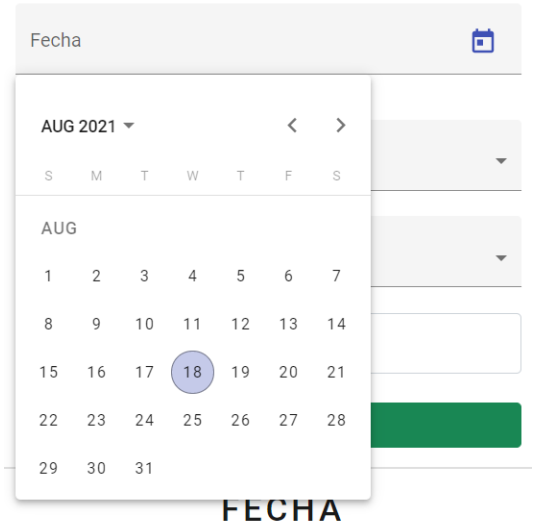
Campo de texto



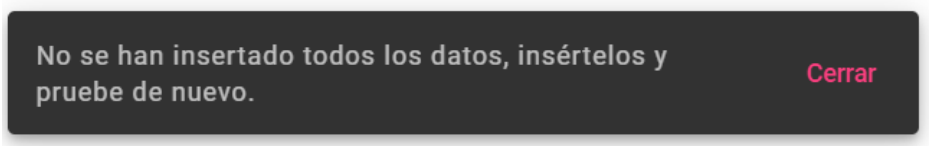
Campo de números



Campo con menú



Campo de fecha



Mensaje de error "No se han insertado todos los datos"

Ilustración 40. Tipos de campo

- **De texto:** (*Campo de texto*) cuadro de texto para las columnas de tipo *varchar* de la base de datos. La cantidad de texto que admite depende del número de caracteres estipulado en la base de datos, que se basa en el tipo de información que se va a guardar. Por ejemplo, el límite es mayor en un campo que almacena descripciones que en uno de nombres.

- **De números:** (*Campo de números*) muy similar al anterior, pero en este caso almacena números. Pueden ser enteros o decimales, igual que sucedía en el campo de texto, esto va a depender de lo que se desee almacenar. Por ejemplo, para añadir el número de elementos que va a realizar un trabajador, éste será un número entero, pero si quiere añadirse el tiempo que se tarda en realizar una tarea se permiten decimales. No se permite introducir números negativos.

- **De menú:** (*Campo con menú*) se muestra en los casos en los que el campo va vinculado a través de una clave externa a otra tabla, como por ejemplo un trabajador y su encargado. En esos casos los únicos valores que se permiten son los que existan en la segunda tabla, por lo que se ha optado por mostrar un desplegable que muestre las opciones posibles, para mayor comodidad del que introduce el dato y para evitar errores. Al abrir la página de inserción de datos, se realiza una consulta a las tablas que aparecen en las opciones del menú, por lo que no habría problema si se manipulan añadiendo, eliminando o modificando información. Por ejemplo, cuando se va a añadir o editar un trabajador, a su vez se realiza una consulta a la tabla de encargados y se muestra la columna deseada en las opciones, de forma que siempre está actualizado. Aunque por pantalla se muestre el nombre, en este caso del encargado, en realidad se está enviando y almacenando el *id* de ese elemento.

- **De fecha:** (*Campo de fecha*) se utiliza en los campos en los que únicamente se aceptan fechas. Permite tanto escribirla a mano en la zona de texto como seleccionarla en el calendario.

- **Error:** (*Mensaje de error "No se han insertado todos los datos"*) en caso de que se deje alguno de los campos vacío, salta el mensaje de error mostrado y no se permite guardar o actualizar los cambios hasta que se introduzca la información.

4.4.4 Personal

Incluye dos ventanas; la de encargados y la de trabajadores. Se trata de las dos únicas tablas que almacenan datos del personal de la fábrica.

4.4.4.1 Encargados

Ilustración 41. Encargados

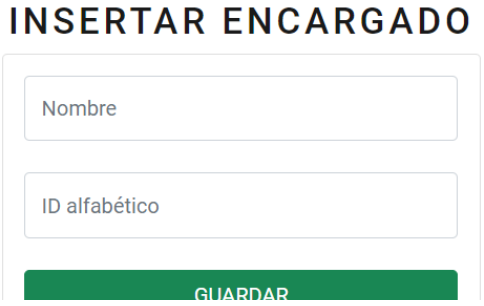


Ilustración 42. Insertar encargado

Se muestran las dos columnas relevantes de la tabla de encargados, es decir, su nombre y su id alfabético. Para añadir uno nuevo se solicitan los dos mismos datos que se muestran.

4.4.4.2 Trabajadores

Ilustración 43. Trabajadores

Ilustración 44. Insertar trabajador

Sólo recoge tres datos, el nombre del trabajador, el turno al que pertenece y el id alfabético de cada uno. Se hace referencia a cada turno por el nombre del encargado que lo dirige. Como ya se ha explicado con anterioridad, si se intentara asignar un turno no existente habría un error, ya que esta tabla está directamente vinculada con la de encargados. Para evitar este error se muestra un desplegable con las posibles opciones que hay en ese momento, es decir, los dos encargados. Si hubiera cambios en la tabla, también se verían reflejados aquí.

4.4.5 Reparto

Aquí se muestran las tablas que no contienen personal de la empresa ni se ocupan del trabajo directamente. Estas son: puestos, referencias, experiencia, índice de incidencias e incidencias.

4.4.5.1 Puestos

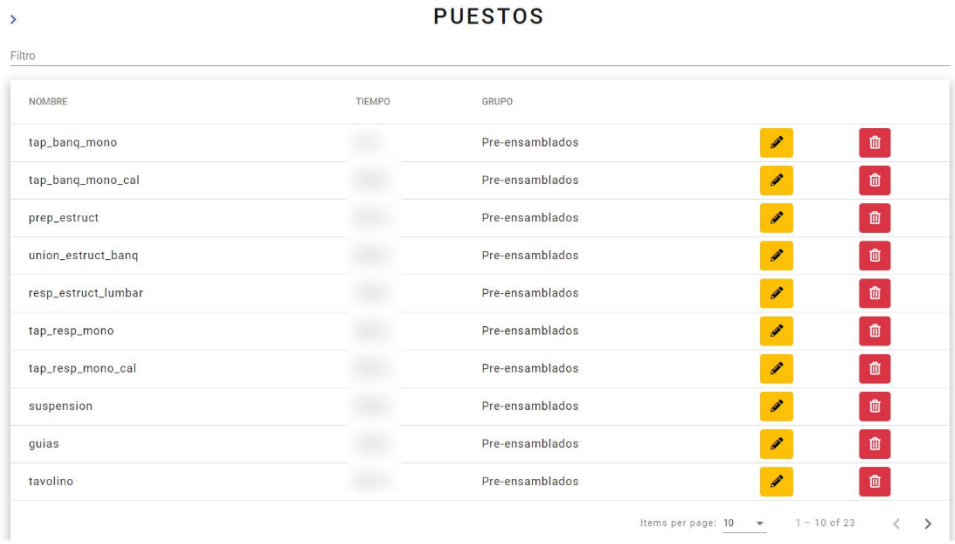


Ilustración 45. Puestos

Ilustración 46. Insertar puesto

Se muestra la lista de puestos y el formulario para añadir uno nuevo o editar uno ya existente.

4.4.5.2 *Referencias*

Ilustración 47. Referencias

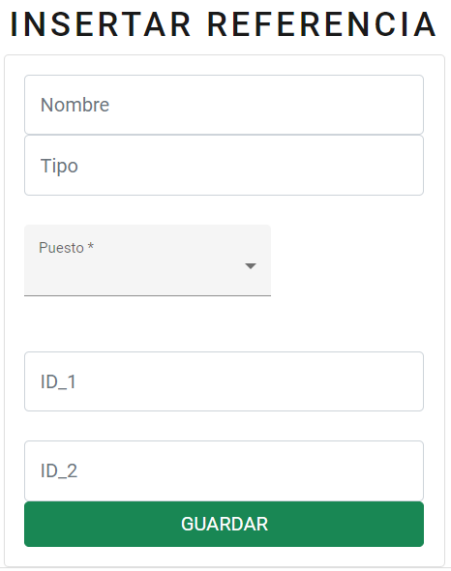


Ilustración 48. Insertar referencia

Se muestra la tabla de referencias, que tiene la o las referencias de cada tipo de asiento, además del panel para poder añadir nuevas referencias. Esta ventana no tiene activado el mensaje de error en caso de que no se introduzca el *id2*, ya que no siempre se tienen dos referencias.

4.4.5.3 *Experiencia*

Ilustración 49. Experiencia

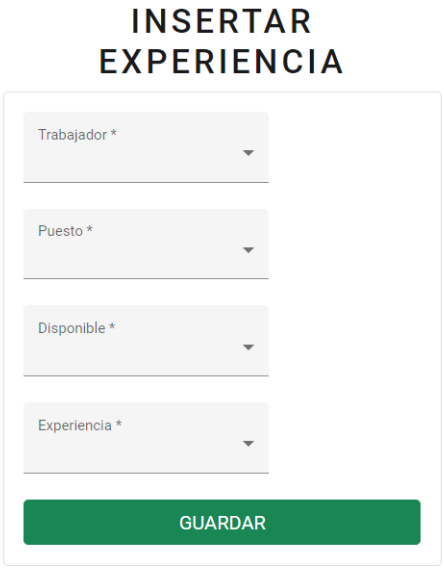


Ilustración 50. Insertar experiencia

Información de cada operario en cada puesto, indicando si está disponible para desempeñarlo o no y su experiencia en caso afirmativo. También contiene un panel únicamente formado por menús desplegables, dado que tanto el trabajador como el puesto tienen que ser uno de los existentes, la disponibilidad sólo puede ser afirmativa o negativa y la experiencia sólo puede estar entre cero y tres.

4.4.5.4 *Índice de incidencias*

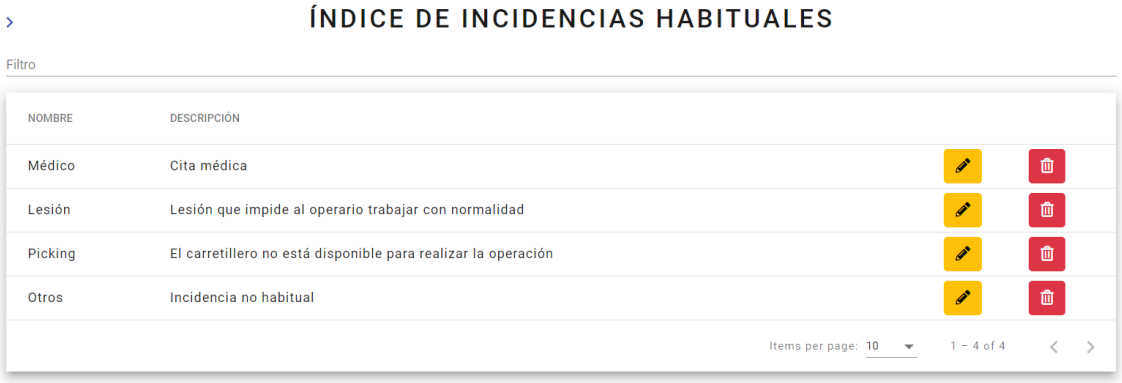


Ilustración 51. Índice de incidencias

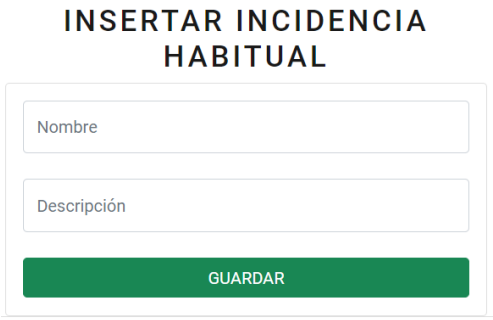


Ilustración 52. Insertar incidencia habitual

Tipos de incidencia que se van a dar de forma habitual, además del cuadro para añadir una nueva. Los datos que se muestran en la *Ilustración 51* son ficticios.

4.4.5.5 *Incidencias*

Ilustración 53. Incidencias

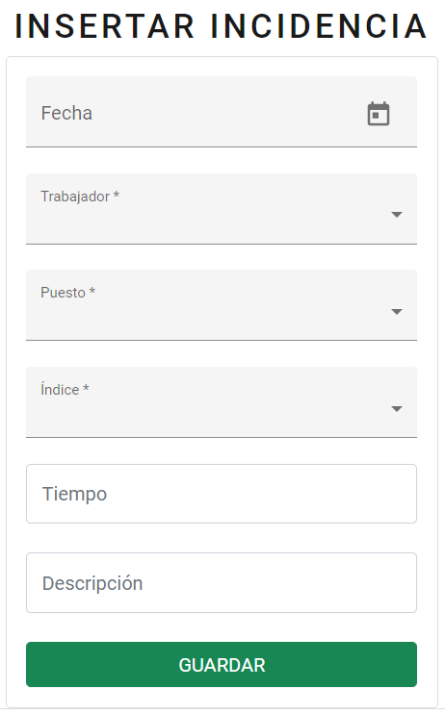


Ilustración 54. Insertar incidencia

Incidencias que han ocurrido durante la jornada. Para registrar una nueva ha de introducirse la fecha en la que se produce dicha incidencia, el trabajador que la sufre y dónde, qué tipo de incidencia es, el tiempo que ha tenido al operario sin poder desempeñar su trabajo y una leve descripción de qué ha sucedido.

4.4.6 Trabajo

Igual que las dos partes anteriores eran muy similares entre sí, esta última tiene muchas diferencias. La complejidad Consta de tres tablas: planificación, registros y registros conjuntos.

4.4.6.1 Planificación

Ilustración 55. Planificación

Ilustración 56. Insertar planificación

Esta pestaña, mostrada en la *Ilustración 55*, alberga multitud de diferencias respecto a todo lo visto hasta ahora.

La primera de ellas es que anteriormente en cada ventana se mostraba cada tabla igual que está en la base de datos o con diferencias mínimas. En este caso es totalmente distinto. Esta tabla en SQL, como ya se vio en el capítulo *4.1.10* únicamente contiene la fecha, el id del trabajador y del puesto y el número de elementos que tiene programados para fabricar. Esta forma de colocar la información es muy útil para organizar la base de datos, pero muy poco práctica para el uso diario del encargado, dado que tendría demasiadas filas y poco organizadas, resultando mucho más cómodo tener a cada empleado en una fila distinta y cada puesto en una columna, pudiendo así ver de un vistazo todo el trabajo asignado a cada trabajador.

Por ello en el *controller* se ha insertado una instrucción más compleja de lo habitual utilizando el comando *PIVOT*, consiguiendo de esta forma que la columna de los puestos pivote y que cada uno sea una nueva columna. Esta instrucción genera muchos campos nuevos. Si un trabajador tiene asignado un solo puesto, al pivotar tendría todos los campos con *NULL* menos el ya existente que seguiría con el mismo valor. Para evitar que esto suceda se ha utilizado el comando *ISNULL*, que hace que se muestren todos los campos *NULL* con el valor que se desee, en este caso el cero, que es en realidad lo que tienen asignado en ese momento, es decir, nada. La consulta de la que se habla se muestra en la *Ilustración 57*.

fecha	trabajador	tap_banq_mono	tap_banq_mono_cal	prep_estruct	union_estruct_banq	resp_estruct_lumbar	tap_resp_mono	tap_resp_mono_cal	suspension	guias	tavolino	tap_banq_bip	bip_tav	bip	final_4w_8w_susp
2021-07-26	[illegible]	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2021-07-26	[illegible]	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2021-07-26	[illegible]	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2021-07-26	[illegible]	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2021-07-26	[illegible]	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2021-07-26	[illegible]	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2021-07-26	[illegible]	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2021-07-26	[illegible]	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2021-07-26	[illegible]	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2021-07-26	[illegible]	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2021-07-26	[illegible]	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2021-07-26	[illegible]	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2021-07-26	[illegible]	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2021-07-26	[illegible]	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2021-07-26	[illegible]	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2021-07-26	[illegible]	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2021-07-26	[illegible]	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2021-07-26	[illegible]	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2021-07-26	[illegible]	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2021-07-26	[illegible]	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2021-07-26	[illegible]	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2021-07-26	[illegible]	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2021-07-26	[illegible]	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2021-07-26	[illegible]	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2021-07-26	[illegible]	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2021-07-26	[illegible]	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2021-07-26	[illegible]	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2021-07-26	[illegible]	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Ilustración 57. Pivot de la planificación en SQL.

Otra de las principales diferencias es el sistema de introducción de datos. Pese a tener una pestaña para añadir una nueva fila a la base, como se puede ver en la *Ilustración 56*, la tabla ha sido configurada para que sea editable y almacene el registro que se introduce. En este caso, cuando se cambia uno de los campos que muestra la tabla, ésta en primer lugar hace una consulta a la base de datos y comprueba si se trata de un dato nuevo o de uno ya existente que está siendo modificado. Esta particularidad se debe a lo explicado con anterioridad acerca de la nueva distribución de la tabla, que muestra ceros donde en realidad la fila no existe. Si tras realizar la comprobación se obtiene que el dato no existía anteriormente, se ejecuta la función de tipo *POST*, mientras que si se encuentra que el registro ya estaba presente, la función que se ejecuta es de tipo *PUT*, es decir, de edición. Debido a ello y a la nueva distribución, esta tabla no cuenta con los botones de editar y eliminar que sí que tienen todas las demás vistas hasta ahora.

El hecho de que se introduzca la información de forma manual en lugar de a través de la pestaña "Insertar planificación", genera otro inconveniente, y es que al abrir la página únicamente se lee la información que ya existe, pero no la que se quiere introducir. Esto es que, si un trabajador tiene asignada una sola tarea en cualquier puesto, ya es suficiente para que aparezca la fila completa con todos los valores a cero, pero para llegar a ello es necesario introducir un primer valor. Esto se ha solucionado añadiendo dos botones, uno para cada encargado, que crea un nuevo registro para cada trabajador en un puesto y en la fecha del día que se aprieta el botón. De esta forma ya se tendría ese primer valor que permite mostrar la fila completa y modificarla según las necesidades. Estos botones también asignan 450 minutos de tiempo disponible a cada trabajador en la tabla registros conjuntos. La diferencia entre el botón de cada encargado es que cada uno añade un registro a los operarios de su turno y no del otro.

Para ampliar la información con la que cuenta el encargado a la hora de asignar trabajo y cantidades, se han añadido dos columnas más que no pertenecen a la planificación: la de productividad y tiempo disponible. Ambas provienen de la tabla de registros conjuntos. Por defecto, como ya se ha explicado, cada vez que se crea un nuevo día se asigna un tiempo disponible a cada trabajador de 450 minutos. Este dato es modificable igual que el resto, a través de la propia tabla.

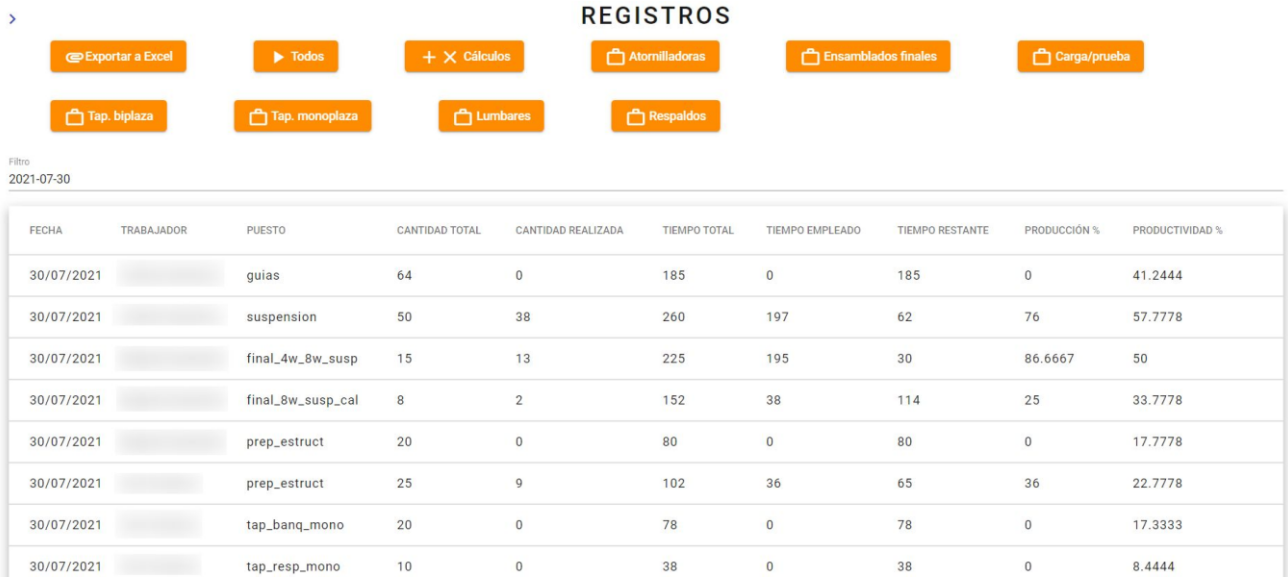
La otra particularidad es la columna de la productividad, que va cambiando de forma automática a medida que se va introduciendo la cantidad de trabajo a cada operario. Esto es muy importante, ya que de esta forma el encargado sabe exactamente si la carga de trabajo es adecuada o no según va asignando el trabajo. Cómo se calcula la productividad está explicado en el capítulo *4.1.12*. En la *Ilustración 58* se muestra un pequeño ejemplo de la planificación, con la productividad calculada automáticamente por el trabajo asignado. No se muestra la fila completa por privacidad de la empresa.

FECHA	TRABAJADOR	PRODUCTIVIDAD %	TIEMPO DISPONIBLE	TAP BANQ MONO	TAP BANQ MONO CAL
20/08/2021	[redacted]	95.9259	450	20	40

Ilustración 58. Ejemplo de planificación

Por último, si se deseara descargar los datos a un Excel, sería tan sencillo como pulsar "Exportar a Excel", botón visible en la *Ilustración 55*.

4.4.6.2 Registros



FECHA	TRABAJADOR	PUESTO	CANTIDAD TOTAL	CANTIDAD REALIZADA	TIEMPO TOTAL	TIEMPO EMPLEADO	TIEMPO RESTANTE	PRODUCCIÓN %	PRODUCTIVIDAD %
30/07/2021	[redacted]	guias	64	0	185	0	185	0	41.2444
30/07/2021	[redacted]	suspension	50	38	260	197	62	76	57.7778
30/07/2021	[redacted]	final_4w_8w_susp	15	13	225	195	30	86.6667	50
30/07/2021	[redacted]	final_8w_susp_cal	8	2	152	38	114	25	33.7778
30/07/2021	[redacted]	prep_estruct	20	0	80	0	80	0	17.7778
30/07/2021	[redacted]	prep_estruct	25	9	102	36	65	36	22.7778
30/07/2021	[redacted]	tap_banq_mono	20	0	78	0	78	0	17.3333
30/07/2021	[redacted]	tap_resp_mono	10	0	38	0	38	0	8.4444

Ilustración 59. Registros

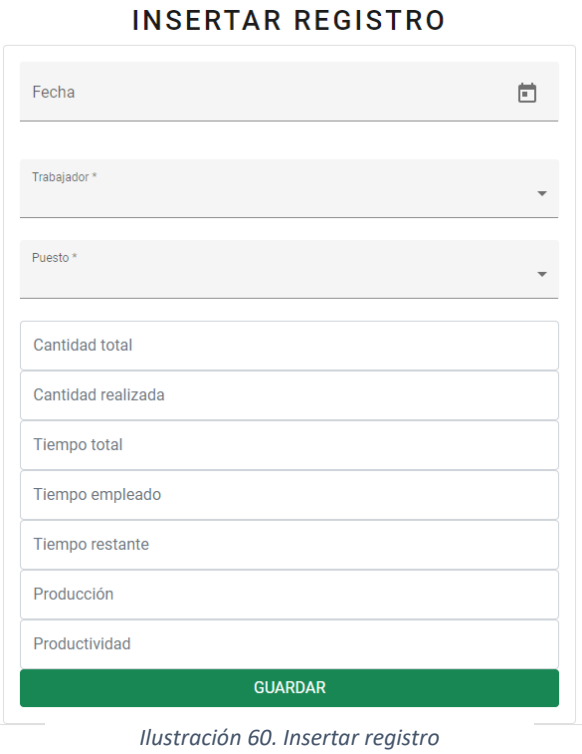


Ilustración 60. Insertar registro

La *Ilustración 59* muestra el trabajo que ha realizado cada trabajador en cada puesto en tiempo real. En dicha imagen, para proteger la privacidad de la empresa, tanto la planificación como los tiempos que se han utilizado para realizar las estadísticas son falsos. Cuenta con diez botones y la tabla en la que se muestra la información. A continuación se explica el funcionamiento de cada uno de los botones:

- **Exportar a Excel:** guarda en un Excel toda la información que se muestra en la tabla (*Ilustración 61*).

fecha	trabajador	puesto	cantidad_total	cantidad_realizada	tiempo_total	tiempo_empleado	tiempo_restante	produccion	productivida
2021-07-30		guias	64	0	185	0	185	0	41,2444
2021-07-30		suspension	50	38	260	197	62	76	57,7778
2021-07-30		final_4w_8w_susp	15	13	225	195	30	86,6667	50
2021-07-30		final_8w_susp_cal	8	2	152	38	114	25	33,7778
2021-07-30		prep_estruct	20	0	80	0	80	0	17,7778
2021-07-30		prep_estruct	25	9	102	36	65	36	22,7778
2021-07-30		tap_banq_mono	20	0	78	0	78	0	17,3333
2021-07-30		tap_resp_mono	10	0	38	0	38	0	8,4444
2021-07-30		tavolino	40	40	232	232	0	100	51,5555
2021-07-30		final_4w_8w_susp	10	6	150	90	60	60	40
2021-07-30		final_8w_susp_cal	7	3	133	57	76	42,8571	35,4667
2021-07-30		resp_estruct_lumbar	44	44	92	92	0	100	24,64
2021-07-30		carga_prueba	33	17	221	113	107	51,5151	49,1333
2021-07-30		tap_banq_bip	55	35	220	140	80	63,6363	48,8888
2021-07-30		bip	14	9	225	144	48	64,2857	50,0888
2021-07-30		bip_tav	9	6	193	129	64	66,6667	43
2021-07-30		union_estruct_banq	12	0	36	0	36	0	8

Ilustración 61. Registros exportados a Excel

- **Todos:** ejecuta de forma simultánea todos los botones excepto el de “Exportar a Excel” y el de “Cálculos”, es decir, todos los que calculan el trabajo realizado en cada puesto.

- **Cálculos:** también afecta a la tabla “Registros conjuntos”. Realiza todos los cálculos estadísticos, esto es, para esta tabla, tiempo total, tiempo empleado, tiempo restante, producción y productividad. Para “Registros conjuntos” son todas las columnas, exceptuando la de “tiempo disponible” que se crea y edita desde la planificación.
- **Atornilladoras:** calcula todo el trabajo realizado desde puestos que contengan una atornilladora. Estas máquinas registran cada apriete en una base de datos SQL de la empresa. Cada vez que se realiza uno, se crea una nueva fila en la tabla, guardando toda la información sobre él. Se almacenan multitud de datos, pero para el propósito perseguido en este trabajo únicamente interesan los explicados posteriormente. El funcionamiento se basa en un bucle que ejecuta una misma consulta variando los trabajadores y los puestos, de forma que se comprueban todas las combinaciones. Esta consulta principalmente lo que hace es contar las distintas etiquetas (utilizando *COUNT* y *DISTINCT*) que se han escaneado añadiendo los filtros (*WHERE*) que se explican a continuación. Con ello puede saberse cuántos elementos ha fabricado cada trabajador en cada puesto ese día.

  Fecha: la máquina guarda la fecha y hora del instante en el que se realiza cada apriete. Solamente se quieren contar las realizadas hoy, ya que las de los días anteriores ya se registraron en su momento. Para ello se han escrito unas líneas de código en la consulta de SQL que almacenan en dos variables la fecha del día en el que se está realizando la consulta, una con la hora del comienzo del turno y otra con la hora del final. De esta forma se puede especificar que solamente interesan los elementos fabricados en esa fecha y a una hora que se encuentre entre el comienzo y el fin del turno.

  Etiqueta: es el dato que se va a utilizar para contar. Todas las piezas tienen una etiqueta que las identifica y que el operario escanea. Tal y como vemos en la tabla *Puestos físicos,* en función del elemento que se esté fabricando, esta etiqueta se almacena en una columna u otra. A la consulta se le pide que cuente las etiquetas del tipo indicado que sean distintas. El tipo indicado es para seleccionar la columna donde están los códigos de las etiquetas que interesan (columna “etiqueta” de la tabla “Puestos físicos”) y que sean distintas es porque, al crear una nueva fila por cada apriete, una misma pieza aparece más de una vez en la base de datos. Si, por ejemplo, se realizan 4 atornillados, este elemento aparece registrado 4 veces, por lo que se especifica que las que sean iguales solamente se cuenten como una. Este punto es el principal de la consulta, el que dice cuánto se ha fabricado, los demás son filtros para concretar quién, dónde y cuándo.

  Dirección IP: gracias a ella se sabe qué puesto es el que se está contabilizando. Dar la etiqueta no es suficiente, ya que en algunos aparecen algunas que no son trascendentales. Por ejemplo, cuando se fabrica un asiento biplaza con *tavolino*, además del código del asiento, también se registra el código del *tavolino*. Si no se diera la dirección IP del puesto, a la persona que estuviera fabricando los citados asientos biplaza también se le

asignaría la producción de los *tavolinos*. Esto no solo sucede con este puesto.

Nombre del trabajador: cada operario tiene una etiqueta impresa que contiene su nombre. Esta etiqueta es escaneada y queda registrado quién ha realizado cada apriete. Al contar los asientos se utiliza esta columna para diferenciar quién ha fabricado cada elemento.

- **Ensamblados finales:** la información se obtiene de los datos que generan las atornilladoras, por lo que la idea general es muy similar a la explicada pero con una dificultad añadida. En este caso los trabajadores pueden realizar tres puestos o tareas desde varios lugares diferentes. Ello implica que, al contrario de lo que sucedía antes, que se comprobaba cada puesto uno a uno conociendo la dirección IP en la que se realizaba, ahora no se puede hacer. Datos que se especifican en la consulta para poder realizar el conteo:

  Fecha: para obtener únicamente los datos generados en la fecha que se busca. El sistema es el mismo utilizado anteriormente, se especifica que el dato se encuentre entre las dos variables de fecha y hora generados previamente.

  Etiqueta del asiento: igual que antes, se especifica que cuente las que sean distintas, excluyendo las pruebas diarias. Se utiliza igual que se hacía antes pero de forma más simple. Estos puestos, denominados como ensamblado final y pertenecientes a los grupos de biplaza y conductor, tienen una columna en la base de datos que únicamente se rellena cuando se trabaja en ellos, por lo que ahora no es necesario introducir una variable que lo indique como ocurría anteriormente, basta con ponerla constante.

  Etiqueta con la referencia: utilizar la etiqueta del asiento es correcto, pero no especifica de qué tipo es. Esto es importante porque no todos los monoplazas son iguales ni tienen el mismo tiempo de fabricación, igual que sucede con los asientos biplaza. Para solucionarlo se realiza una consulta dentro de la original, que accede a la tabla de *Referencias de los asientos* y comprueba las referencias. Lo que se consigue es poder diferenciar el asiento que se está fabricando, independientemente del tipo que sea.

  Dirección IP: para determinar el puesto en el que se ha fabricado el asiento.

  Nombre del trabajador: se usa exactamente de la misma forma y para el mismo propósito que en “Atornilladoras”, para saber quién ha fabricado el elemento.

- **Carga/prueba:** se trata del puesto que realiza las comprobaciones finales a los asientos que salen a la fábrica. Cada asiento que se escanea se registra en una base de datos de SQL, por lo que se accede a ella para comprobar la información. Se utiliza un bucle que ejecuta una consulta con el nombre de cada operario. Esta consulta comprueba y cuenta los distintos números de serie que se escanean en el puesto el día que se está trabajando por cada trabajador, sabiendo así los asientos que ha probado cada uno.

- **Tapizado de asientos biplaza:** igual que carga/prueba, funciona con un bucle que introduce el nombre de cada persona en una consulta que cuenta las diferentes etiquetas que se han escaneado en ese puesto, sabiendo así el número exacto de tapizados que ha realizado cada trabajador en ese puesto en ese día.
- **Tapizado de asientos monoplaza:** funciona de forma exactamente igual que el de tapizados biplaza, siendo el código casi idéntico. La diferencia es que esta consulta se realiza sobre otra tabla que se encuentra en un ordenador distinto, lo que obliga a duplicar la función.
- **Lumbares:** también muy similar a las dos funciones de los tapizados, pero ligeramente más sencillo. En este caso sólo se crea un registro por cada pieza que se fabrica, por lo que simplemente hay que contar el número de veces que aparece el nombre del operario en la fecha indicada.
- **Respaldos:** los datos que se registran cuando se fabrica un respaldo a priori son muy simples, se crea una nueva fila por cada respaldo con la fecha y el nombre de cada operario, por lo que contando el número de veces que aparece el trabajador debería bastar. El problema es que los hay calefactados y no calefactados, por lo que para diferenciarlos también hay que utilizar la referencia del asiento. Lo que se recibe del programa es el nombre del trabajador, que ya se ha especificado cómo se utiliza, y el nombre del puesto. Para llegar desde la tabla de respaldos hasta el puesto al que corresponde se hace una sucesión de consultas, una dentro de otra. Dentro de la original se introduce una que comprueba a qué referencia corresponde la que se tiene en la tabla de respaldos y desde ella se añade otra que comprueba a qué puesto pertenece la referencia. Si la referencia es de un puesto en el que se fabrican asientos no calefactados, el respaldo tampoco lo será, si es de un puesto en el que sí lo son, el respaldo también. De esta forma se conoce el número de respaldos calefactados y no calefactados que ha fabricado cada trabajador.

4.4.6.3 Registros conjuntos

Ilustración 62. Registros conjuntos

En esta ventana, visible en la *Ilustración 62*, se encuentra toda la información almacenada en la tabla de SQL “Registros conjuntos”. Se muestra tal cual está guardada, salvo por la columna de trabajadores que en lugar de tener los *id* de cada trabajador tiene directamente su nombre. Se trata de una recopilación global de todo lo mostrado en la tabla “Registros”, donde se encuentra la información desglosada por puestos. Aquí se presenta una fila por cada trabajador. Al igual que la otra tabla, también cuenta con un botón para exportar a Excel los resultados que se muestran.

5. Conclusiones

En este capítulo se hablará acerca de las conclusiones a las que se ha llegado y de las posibles líneas futuras que tomará el proyecto.

5.1 Conclusiones

El proyecto que se ha llevado a cabo ha servido para desarrollar un sistema de recogida de datos de diferentes fuentes y formatos, unificándolos y facilitando su acceso y comprensión, ayudando tanto a trabajadores como a encargados y directivos en su trabajo en el día a día. Los beneficios resultarán inmediatos y se tomarán decisiones basadas en datos recogidos automáticamente. Los operarios no necesitarán anotar manualmente el trabajo realizado durante la jornada, pudiendo así aprovechar este tiempo para realizar otras tareas o simplemente para ir más desahogados durante el turno, ya que como se explicaba al principio del documento, uno de los problemas a los que se enfrenta la empresa es el ritmo excesivo que tratan de seguir y que a largo plazo desencadena lesiones. Además, gracias a la integración de la aplicación en el sistema y el hecho de que funcione con los registros generados por las máquinas en el taller, no existe el error en la contabilidad de los elementos fabricados, a diferencia de lo que ocurre ahora, que al realizarla los operarios de forma manual mientras trabajan, los errores son comunes. Para los encargados la principal ventaja es que contarán con una herramienta que les informe de la situación de cada trabajador en todo momento, además del nuevo sistema de planificación, mucho más rápido y moderno que el anterior, que les permitirá ahorrar tiempo y crear un registro más eficiente. Por último también será útil para la dirección, donde ayudará al jefe de logística, interesado en los asientos que finalmente abandonan la fábrica, que pasará a tener esa información mucho más accesible y precisa; y a la jefa de producción y gerente de la empresa, que tendrá un control mucho mayor de todo lo que suceda en el taller, además de contar con registros para acceder a información anterior.

La culminación de éste es su instalación final e implantación en la actividad diaria de la empresa, dejando de lado algunas de las rutinas actuales que se volverán innecesarias gracias a esta herramienta.

Ha sido muy importante para el desarrollo el haber trabajado siguiendo el método Scrum, que ha permitido seguir un sistema claro y eficaz, teniendo siempre claros los objetivos próximos sin perderse al intentar abarcar una idea demasiado global.

Por último, es importante destacar que esto ha sido posible en parte gracias a lo aprendido en el Grado en Tecnologías Industriales en la Universidad de Valladolid, donde he adquirido las competencias necesarias para poder llevar a efecto este proyecto.

5.2 Líneas futuras

El objetivo más próximo es su correcta implantación y adecuar a los trabajadores a su uso, especialmente a los encargados, que serán los que más cambios noten en su día a día.

Una de las mejoras que ya se está pensando en instaurar es la implantación de la herramienta de Angular denominada Socket.io, que actualizaría los registros de forma periódica sin necesidad de presionar botones para ello. Con ello se lograría un ahorrar tiempo y proporcionar comodidad al empleado que la utilice, ya que la aplicación se mantendría con datos actualizados de forma constante. Esto ya está planteado y el programa tiene sentadas las bases para que se lleve a cabo.

Definiciones

API: (Application Programming Interface) "*es la plataforma que utiliza un programa para acceder a diferentes servicios en el sistema informático*" (Ferrandis, 2018).

Backend*: "parte de una aplicación que almacena y manipula datos*" (Ferrandis, 2018).

Booleano: *"tipo de dato que contiene valores que sólo pueden ser True o False"* (Microsoft, 2015). En MSSQL este tipo de dato se denomina *BIT*, siendo 1 verdadero y 0 falso (Microsoft, 2017).

Clave principal: también conocido como *primary key*, es un tipo de restricción única de SQL. El objetivo de este tipo de clave es hacer única a cada fila de la tabla, de forma que ninguna esté nunca duplicada. No admite valores nulos y sólo puede definirse una para cada tabla. (Oppel & Sheldon, 2010)

Clave externa: también conocido como *foreign key.* Esta restricción "*se ocupa de cómo los datos en una tabla hacen referencia a los datos en otra tabla*" (Oppel & Sheldon, 2010).

Entorno de ejecución: se trata de la infraestructura de hardware y software que admite la ejecución de una base de código en particular en tiempo real. Incluye el tipo de CPU, el sistema operativo y cualquier motor de tiempo de ejecución o software del sistema requerido por una categoría particular de aplicaciones.

Evento asíncrono: aquel tipo de evento que permite que la ejecución de éstos pueda ocurrir en paralelo mientras la máquina virtual continúa con su ejecución.

Frontend*: "interfaz a través de la cual los usuarios comunes pueden acceder a un programa"* (Ferrandis, 2018).

Inferencia de tipos: "*asignar a cada expresión de un programa su tipo correspondiente*" (Montenegro Montes, 2007).

Iteración: "*acción y efecto de repetir*" (Real Academia Española, 2020).

Superconjunto: "*incluye todas las funcionalidades del original pero añade funcionalidades extra propias*" (Rospigliosi, 2020).

Bibliografía

Angular. (s. f.). *Introduction to services and dependency injection*. Recuperado 22 de julio de 2021, de https://angular.io/guide/architecture-services

Atlassian. (s. f.-a). *Development and Collaboration Software Company*. Recuperado 31 de agosto de 2021, de https://www.atlassian.com/company

Atlassian. (s. f.-b). *Jira Overview*. Recuperado 26 de agosto de 2021, de https://www.atlassian.com/software/jira/guides/getting-started/overview

Atlassian. (s. f.-c). *Scrum*. Recuperado 31 de agosto de 2021, de https://www.atlassian.com/es/agile/scrum

Comunidad TypeScript. (s. f.). *TypeScript*. Recuperado 31 de agosto de 2021, de https://www.typescriptlang.org/

Ferrandis, J. (2018). *Desarrollo back-end en .NET de una aplicación para la obtención de opiniones mediante gamificación* .

Git. (s. f.-a). *About*. Recuperado 1 de septiembre de 2021, de https://git-scm.com/about

Git. (s. f.-b). *Git*. Recuperado 27 de agosto de 2021, de https://git-scm.com/

Git. (s. f.-c). *Git - A Short History of Git*. Recuperado 31 de agosto de 2021, de https://git-scm.com/book/en/v2/Getting-Started-A-Short-History-of-Git

Microsoft. (s. f.). *Access SQL: conceptos básicos, vocabulario y sintaxis - Access*. Recuperado 18 de abril de 2021, de https://support.microsoft.com/es-es/office/access-sql-conceptos-básicos-vocabulario-y-sintaxis-444d0303-cde1-424e-9a74-e8dc3e460671

Microsoft. (2015). *Tipo de datos Boolean*. https://docs.microsoft.com/es-es/dotnet/visual-basic/language-reference/data-types/boolean-data-type

Microsoft. (2017). *bit (Transact-SQL)*. https://docs.microsoft.com/es-es/sql/t-sql/data-types/bit-transact-sql?view=sql-server-ver15

Microsoft. (2018). *IntelliSense in Visual Studio*. https://docs.microsoft.com/en-us/visualstudio/ide/using-intellisense?view=vs-2019

Montenegro Montes, M. (2007). *Inferencia de tipos seguros en un lenguaje funcional con destrucción explícita de memoria*.

Montgomery, J. (2015). *BUILD 2015 News: Visual Studio Code, Visual Studio 2015 RC, Team Foundation Server 2015 RC, Visual Studio 2013 Update 5*. https://devblogs.microsoft.com/visualstudio/build-2015-news-visual-studio-code-visual-studio-2015-rc-team-foundation-server-2015-rc-visual-studio-2013-update-5/

Node.js. (s. f.-a). *Acerca de Node.js*. Recuperado 18 de julio de 2021, de https://nodejs.org/es/about/

Node.js. (s. f.-b). *Introduction to Node.js*. Recuperado 26 de agosto de 2021, de https://nodejs.dev/learn

Oppel, A., & Sheldon, R. (2010). *Fundamentos de SQL*.

Postman. (s. f.). *The Postman API Platform*. Recuperado 18 de abril de 2021, de https://www.postman.com/api-platform/

Postman. (2021). *How We Built Postman - the Product and the Company*. https://blog.postman.com/how-we-built-postman-product-and-company/

Real Academia Española. (2020). *Diccionario de la lengua española*. https://dle.rae.es/iteración

Rospigliosi, P. (2020). *Estudio sobre Angular 2 y superior*. 51. https://zaguan.unizar.es/record/10321/files/TAZ-TFM-2013-144.pdf

Socket.IO. (2021). *Introduction*. https://socket.io/docs/v4

Visual Studio. (s. f.). *Documentation for Visual Studio Code*. Recuperado 18 de abril de 2021, de https://code.visualstudio.com/docs

Visual Studio. (2021). *Code Navigation in Visual Studio Code*. https://code.visualstudio.com/docs/editor/editingevolved