



**Comunidad
de Madrid**



IES Europa

TRABAJO FIN DE GRADO

Grado superior Desarrollo de Aplicaciones Multiplataforma.
Curso académico 2024|2025.

ProLink

Optimización y Control de la Producción en Empresas Industriales.

Autor: Juan Pedro Rodríguez Aranda.

Tutor: Peter Soldado Arcos.

Entrega 31/05/2025

Índice

1. Introducción.....	4
1.1. Justificación.....	4
1.2. Ubicación y destinatarios.....	4
1.3. Antecedentes.....	4
1.4. Integrantes y reparto de funciones.....	4
2. Objetivos específicos.....	5
2.1. Estudio de viabilidad.....	5
2.2. Diagrama casos de uso.....	6
2.3. Diagrama de Gantt.....	6
2.4. Infografía.....	7
2.5. Infografía del proceso.....	7
3. Análisis y diseño del sistema.....	8
3.1. Requisitos del Sistema.....	8
3.2. Modelado de la Base de Datos.....	10
3.3. Backend (Servidor) - Spring Boot.....	11
3.4. Frontend(Cliente) - React.....	19
4. Instalación.....	25
4.1. Instalación de entorno MYSQL.....	26
4.2. Instalación de entorno IntelliJ IDEA.....	30
4.3. Ejecutar nuestro servicio API Rest.....	31
4.4. Ejecutar nuestra interfaz.....	32
5. Resultado.....	33
6. Conclusiones y nuevas propuestas:.....	33
6.1. Ventana de usuarios.....	33
6.2. Sistema de notificaciones.....	33
6.3. Registro de acciones.....	33
6.4. Sistema de búsqueda.....	33
7. Referencias bibliográficas y documentales.....	33

1. Introducción

1.1. Justificación.

La fase de producción de muchas empresas presenta desafíos significativos debido a la falta de herramientas adecuadas para el seguimiento de pedidos y productos en tiempo real. En la actualidad, los empleados deben desplazarse físicamente para obtener información, lo que genera retrasos, descoordinación y problemas de stock.

Este proyecto surge como una solución para automatizar y optimizar la gestión de producción mediante una aplicación web que centraliza toda la información, mejorando la comunicación y eficiencia operativa.

1.2. Ubicación y destinatarios.

Este proyecto está dirigido a empresas de manufactura que buscan mejorar y agilizar su proceso de producción. Se implantará en una Pyme del sector industrial que enfrenta problemas en el proceso productivo.

1.3. Antecedentes

Una de las áreas con mayor necesidad de innovación es la gestión de pedidos en tiempo real, ya que muchas pymes aún dependen de métodos manuales o sistemas poco integrados, lo que limita su eficiencia operativa. Para ello existen los **sistemas MES (Manufacturing Execution Systems)**, mejoran significativamente la trazabilidad y control de la producción.

Aunque muchas de estas soluciones suelen estar enfocadas en grandes industrias y resultan costosas o demasiado complejas para una Pyme. Esta brecha ha impulsado el desarrollo de soluciones más accesibles como **aplicaciones web personalizadas**, diseñadas específicamente para las necesidades y recursos de pequeñas empresas manufactureras.

1.4. Integrantes y reparto de funciones.

Juan Pedro Rodríguez Aranda: Backend, Frontend, documentación.

2. Objetivos específicos.

- **Crear una aplicación web** que centralice la información relevante al proceso de producción de un pedido en una empresa.
- **Multiplataforma** accesible tanto en ordenadores como dispositivos móviles independientemente del SO.
- **Mejorar la comunicación interna** entre los diferentes departamentos de producción.
- **Minimizar el tiempo perdido** en consultas internas sobre el estado de los pedidos.
- **Gestionar pedidos en tiempo real** permitiendo a los trabajadores identificar problemas o retrasos rápidamente.
- **Interfaz intuitiva** para empleados sin experiencia técnica.
- **Desplegar la aplicación en la red interna** de la empresa para mayor seguridad.
- **Sistema de autenticación de usuario** para agregar permisos especiales en base al cargo de trabajo.
- **Implementar notificaciones** para alertar sobre retrasos o problemas en la producción.

2.1. Estudio de viabilidad.

Frontend (Interfaz de Usuario):

- **Framework:** React para desarrollar una aplicación web responsiva.
- **Estilos:** TailwindCSS para diseño atractivo y rápido.

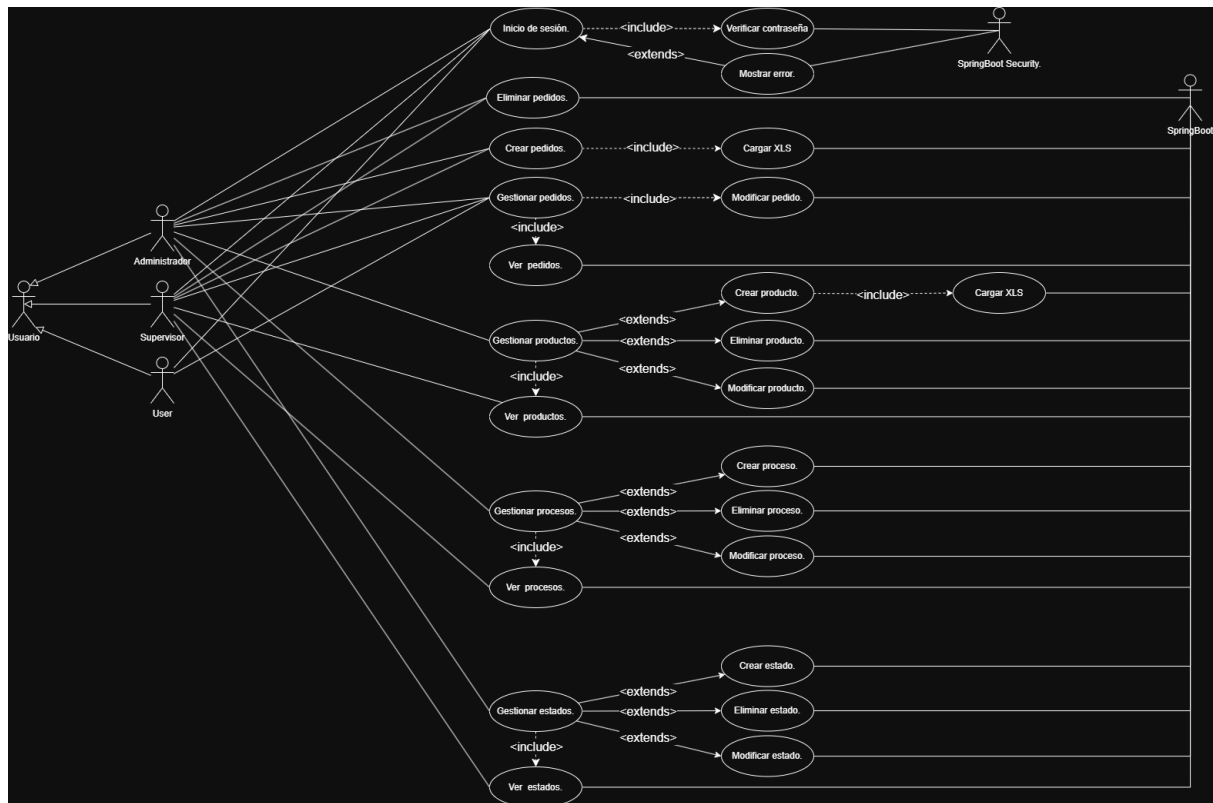
Backend (Lógica y Datos):

- **Lenguaje:** Java con Spring Boot para construir una API REST.
- **Base de Datos:** MySQL.

Herramientas de Desarrollo:

- **Control de versiones:** Git para gestionar el código, Flyway para versionado de base de datos y SonarLint para gestionar métricas del proyecto.
- **Entorno de desarrollo:** IntelliJ IDEA.
- **Despliegue:** Red local para mayor seguridad.

2.2. Diagrama casos de uso.



2.3. Diagrama de Gantt.

	Marzo	Abril	Mayo	Junio
Análisis y Planificación	Diseño de la arquitectura Creación Mockups			
Desarrollo Backend	Configurar del entorno Spring Boot y MySQL Desarrollo Backend y API REST			
Desarrollo Frontend		Configuración de React y TailwindCSS Implementación de interfaz e integración con Backend		
Pruebas Finales			Despliegue en red local Pruebas finales Corrección de errores y mejoras	
Documentación	Redacción de memoria TFG			
Preparación de defensa TFG				Ensayo de la defensa del TFG

2.4. Infografía.



2.5. Infografía del proceso.

1. Un usuario se registra, si las credenciales son correctas accede, si no, se mostrará un error.
2. Un administrador o supervisor creará una venta o pedido con un parte de fabricación en hoja de excel (xls).
3. Los usuarios actualizan el estado de las unidades de producción.
4. El supervisor controlará este flujo de trabajo.
5. Si un producto resulta dañado el supervisor devolverá la unidad de producción al estado correspondiente y estados posteriores aparecerán en rojo.
6. Se completará el flujo normal hasta la entrega del producto.
7. Si un producto ha sido entregado con éxito un administrador o supervisor eliminará la venta marcando así el fin.
8. Si un producto requiere un proceso nuevo o estados nuevos, será el administrador quien los agregue.
9. Si un producto tiene un proceso erróneo o un proceso necesite actualizar sus estados, será el supervisor quien avise al administrador.

3. Análisis y diseño del sistema.

3.1. Requisitos del Sistema.

Requisitos funcionales:

1. Gestión de usuarios

- Los usuarios deben poder loguearse en el sistema.
- El sistema permite la autenticación con usuario y contraseña.
- Los usuarios tendrán roles con diferentes permisos.

2. Gestión de pedidos.

- Los administradores y supervisores pueden crear y eliminar pedidos.
- Los administradores, supervisores y usuarios básicos pueden ver y modificar el estado de un pedido.

3. Gestión de productos.

- Los administradores pueden crear, modificar y eliminar productos.
- Los administradores y supervisores pueden ver los productos.

4. Gestión de procesos

- Los administradores y supervisores pueden ver los procesos.
- Solo los administradores podrán crear, modificar o eliminar procesos.

5. Gestión de estados.

- Los administradores y supervisores pueden ver los estados.
- Solo los administradores podrán crear, modificar o eliminar estados.

6. Accesibilidad y Seguridad

- La aplicación debe ser multiplataforma y responsiva para dispositivos móviles y ordenadores.
- Se debe restringir el acceso a ciertos módulos según el rol del usuario.
- La aplicación debe garantizar seguridad en la autenticación y datos.
- La aplicación verificará los usuarios mediante tokens JWT.

Requisitos no funcionales:

1. Rendimiento y Eficiencia

- El sistema debe procesar solicitudes de manera asíncrona.
- La base de datos debe manejar los datos de manera atómica.
- La interfaz debe ser rápida y fluida.

2. Seguridad

- La autenticación debe implementarse con Spring Security.
- Solo los usuarios con permisos adecuados pueden modificar datos.
- La comunicación entre frontend y backend será mediante la api y se propagan los cambios por webSocket.

3. Escalabilidad

- El sistema debe permitir la adición de pedidos, productos, procesos y estados sin afectar el rendimiento.

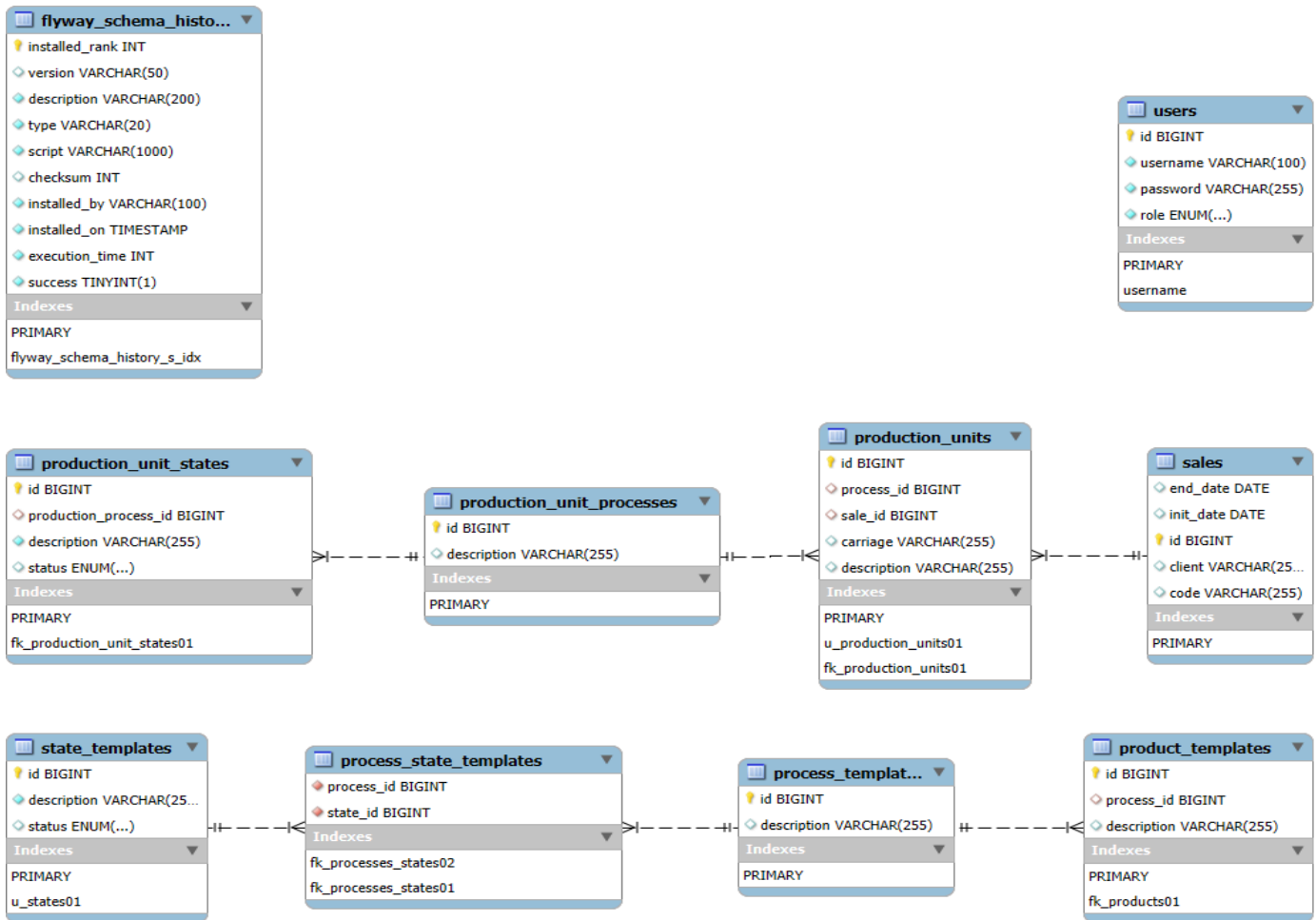
4. Usabilidad y Accesibilidad

- La interfaz debe ser intuitiva y fácil de usar para empleados sin experiencia técnica.
- Debe seguir principios de diseño responsivo para adaptarse a diferentes dispositivos.
- Debe permitir la adición de pedidos o productos mediante XLS.

5. Mantenimiento y Actualización

- El código debe ser modular y bien documentado para facilitar futuras mejoras.
- Las actualizaciones deben poder realizarse sin afectar el funcionamiento de la empresa.
- Se debe garantizar compatibilidad con las versiones actuales de los navegadores.

3.2. Modelado de la Base de Datos.



Este modelo de datos lo podemos dividir en cuatro capas:

1. **flyway_schema_history**: Tabla generada automáticamente por flyway para controlar el versionado de la base de datos. Mantiene un historial detallado de los cambios (scripts), permite mantener la consistencia entre entornos. Gracias a esto podemos reproducir fácilmente su estructura.
2. **Usuarios**: Contiene su nombre y contraseña codificada mediante BCrypt con Spring Security.
3. **Plantillas**: Todas las tablas con la terminación “_templates” sirven como base para generar posteriormente la fase de producción.
4. **Unidades de producción**: Representan la entidad en la fábrica, son elementos únicos, independientes y modificables.

3.3. Backend (Servidor) - Spring Boot.

El proyecto utiliza una **arquitectura basada en microservicios**, en la que cada servicio expone su funcionalidad a través de una **API REST**. Cada servicio está estructurado internamente mediante una **arquitectura en capas**:

1. Resources:

- 1.1. **application.properties**: Define la ruta a la base de datos, establece el driver y activa el uso de flyway.
- 1.2. **db.migration**: Contiene los script sql que lanza flyway para establecer la estructura y comprobar su integridad.

2. Security:

- 2.1. **SecurityConfig**: Permite configurar de manera personalizada la seguridad para el proyecto. Usa Beans, objetos gestionados por el contenedor de Spring, es decir instancias creadas y configuradas por Spring de manera autónoma.
 - Configura **CORS** (Cross-Origin Resource Sharing) permitir peticiones sólo desde ciertos orígenes o dominios, métodos y cabeceras.
 - Desactiva **CSRF** para permitir el uso de **tokens JWT** que el servidor verifica en cada petición y establece que rutas son públicas y cuáles no.

```
.requestMatchers("/api/auth/**").permitAll()
```

```
.requestMatchers("/state/**").authenticated()
```

- Uso de algoritmo **BCryptPasswordEncoder** encriptar y verificar contraseñas.
- 2.2. **JwtUtil**: Utilidad para manejar **tokens JWT** (JSON Web Tokens).
 - Generar tokens JWT, este será válido durante 10 horas.
 - Validar tokens JWT.
 - Extraer el nombre del usuario desde el token.

- 2.3. **JwtAuthenticationFilter**: Autentica a los usuarios usando tokens JWT incluidos en las cabeceras de las **peticiones HTTP**.

- Extiende de **OncePerRequestFilter**, por lo que se ejecuta por cada petición HTTP.

- Extrae el token del header, si es nulo o carece de **Bearer** (Token de acceso que autoriza a quien lo porta) no permite el acceso.
- Valida el token, si no es correcto o ha expirado deniega el acceso.
- Extrae el usuario y verifica el nombre y contraseña.

3. Configurations:

3.1. WebSocketConfig: Configura y habilita Websockets con soporte para **STOMP** (Streaming Text Oriented Messaging Protocol), un protocolo de mensajería simple basado en texto plano, con el que propagaremos los datos en tiempo real a los clientes conectados.

- Activa **MessageBroker** que funciona como intermediario de mensajes.
- Activa un broker en memoria con la ruta **"/topic"** para enviar mensajes
- Registra el **endpoint STOMP** para WebSockets en **"ws"** y limita el acceso a direcciones **localhost como 5173**, usada por react.
- Habilita **SockJs**, una librería que permite usar WebSockets o fallback a otras tecnologías si no están soportadas por el navegador

3.2. DataLoader: Inicializa 3 usuarios básicos, admin, supervisor, user para el campo de pruebas, como las contraseñas son codificadas con **BCryptPasswordEncoder** no pueden ser hardcodeadas en sql y al menos un usuario necesita ser introducido mediante java.

4. Dominio.Modelo:

Representa el **núcleo del negocio**, es decir, **entidades y repositorios**

4.1. Plantillas:

Estas entidades representan información estática, están pensadas para nutrir a las entidades de producción y mantener una estructura de datos.

- **Plantilla de estado:** Estados de producción, puede ser **ninguno**, **en proceso**, **retrasado** o **terminado**.
- **Plantilla de proceso:** Proceso de producción, puede ser **pintura metalizada** depende de **mezclar**, **embotellar** y **repartir**.
- **Plantilla de producto:** Producto, por ejemplo el producto **pintura metalizada roja** tiene **pintura metalizada**.

4.2. Unidades de producción:

Entidades dinámicas presentes en la fabricación de cada producto, usan la información base de las plantillas para poder ser modificadas y desacoplar totalmente su dependencia .

- **Estado de unidad de producción.**
- **Proceso de unidad de producción.**
- **Unidad de producción:** Se trata del producto dentro de la fábrica, la unidad a producir.

4.3. Venta:

Representa **un parte de fabricación o una copia de venta**, con información relevante para el operario, como el código de pedido, el cliente, fecha de inicio, fecha de fin y la lista de productos (como unidad de producción) que se deben producir.

4.4. Usuario:

Entidad que representa los trabajadores de la empresa, tiene nombre y contraseña y están divididos en roles:

- **ADMIN:** Acceso total a la aplicación.
- **SUPERVISOR:** Acceso parcial a la aplicación-
- **USER:** Acceso básico a la aplicación.

5. Dominio.Repositorio:

Interfaces que definen cómo acceder a los datos del dominio, una por cada clase mencionada anteriormente. Descienden de `JpaRepository` **interfaz de Spring Data JPA** que te permite acceder a una base de datos **sin escribir SQL manualmente**. Es parte del framework **Spring Data**, que simplifica el trabajo con **JPA**.

6. Servicios:

Clases que implementan la lógica de aplicación: coordinan llamadas al dominio, acceden a los repositorios, y aplican reglas de negocio **usando las entidades del dominio**. Una por cada repositorio y todas contiene los siguientes métodos:

- **Guardar:** Persistir la entidad en la BBDD.
- **Buscar todos:** Recibir todas las entidades de la BBDD.
- **Encontrar por Id:** Recibir la entidad por identificador de la BBDD.
- **Actualizar:** Actualizar la entidad en la BBDD.
- **Eliminar por Id:** Eliminar la entidad por identificador.

En adición a estos métodos, en el servicio de ventas encontramos

- **ProductTemplateToProductionUnit:** Transforma la plantilla de usuario a unidad de producción.

Y en el servicio de productos:

- **saveAll:** Persiste productos desde una lista, para cargar datos de forma rápida.

7. Objeto de transferencia de datos (DTO):

Clases diseñadas para transportar datos entre capas, **especialmente entre el controlador y el cliente**.

- 7.1. **Solicitud de autenticación (AuthRequest):** Usado como input para iniciar sesión.
- 7.2. **Respuesta de autenticación (AuthResponse):** Usado para devolver el token si el inicio de sesión ha sido exitoso.
- 7.3. **Plantilla de producto DTO:** Usado para transformar los datos de XLS y mapear a unidad de producción. Cuenta con la cantidad de unidades y los componentes de la misma.
- 7.4. **Componentes DTO:** Usado para definir de qué elementos se compone un producto.
- 7.5. **Venta DTO:** Usado para transformar los datos extraídos de una parte de fabricación o una venta XLS.

8. Controladores

Los **controladores REST y WebSocket** son responsables de manejar las solicitudes que llegan desde el cliente y generar las respuestas correspondientes. Actúan como la capa de entrada de nuestra aplicación, exponiendo los **endpoints** de la API y aplicando restricciones de acceso según el **rol del usuario**.

Todos los controladores REST están anotados con:

```
@RestController
@RequestMapping("/nombre-del-controlador")
```

- `@RestController`: Indica que la clase es un controlador REST y que los métodos devuelven datos directamente, en esta ocasión tratamos con estructuras JSON.
- `@RequestMapping`: Define la ruta base para los endpoints definidos dentro de esa clase.

En Spring, los controladores devuelven una **respuesta HTTP completa** utilizando la clase `ResponseEntity<Object>`, lo que permite un control total sobre:

- El **cuerpo** de la respuesta.
- El **código de estado HTTP**.
- Las **cabeceras** si es necesario.

Esto nos proporciona una gran **flexibilidad** para manejar diferentes escenarios dentro del mismo método, como respuestas exitosas, errores de validación o accesos no autorizados. Los códigos más comunes son:

```
OK(200, Series.SUCCESSFUL, "OK")

CREATED(201, Series.SUCCESSFUL, "Created")

ACCEPTED(202, Series.SUCCESSFUL, "Accepted")

NO_CONTENT(204, Series.SUCCESSFUL, "No Content")

BAD_REQUEST(400, Series.CLIENT_ERROR, "Bad Request"),

UNAUTHORIZED(401, Series.CLIENT_ERROR, "Unauthorized"),

NOT_FOUND(404, Series.CLIENT_ERROR, "Not Found"),

CONFLICT(409, Series.CLIENT_ERROR, "Conflict"),

GONE(410, Series.CLIENT_ERROR, "Gone"),
```

Además, la seguridad está gestionada mediante anotaciones como `@PreAuthorize`, que restringen el acceso a los endpoints según los **roles definidos en el sistema** (`ADMIN`, `SUPERVISOR`, `USER`, etc.).

Los controladores WebSocket, por otro lado, permiten la **comunicación en tiempo real** con los clientes mediante el uso de `SimpMessagingTemplate`, permitiendo a los usuarios recibir eventos sin necesidad de hacer polling.

Además, con la anotación `@Slf4j` proporcionada por **Lombok**, se habilita automáticamente un logger en la clase, sin necesidad de instanciarlo manualmente. Esto permite escribir

mensajes de registro de manera sencilla usando métodos como `log.info()`, `log.error()`, etc.

Veamos unos ejemplos de controlador:

8.1. Autenticación (AuthController): Expone el endpoint de **inicio de sesión**. Es **público** y no requiere autenticación previa ya que tiene la función de validar las credenciales del usuario y en caso de ser correctas, **generar y devolver un token JWT** que podrá ser utilizado en solicitudes posteriores para acceder a recursos protegidos.

- **Recibe una solicitud** `AuthRequest` con el nombre de usuario y la contraseña.
- **Autentica al usuario** mediante `AuthenticationManager`.
- Si la autenticación **es exitosa**:
 - **Recupera** el nombre de **usuario** y el **rol**.
 - **Genera un token JWT** que contiene esta información.
 - **Retorna** un objeto `AuthResponse` con el token.
- Si las credenciales son incorrectas, se responde con un código **404 Not Found**.

8.2. Venta (SaleController): Expone los **endpoints** relativos a una **venta**, Tiene habilitada la seguridad a nivel de métodos mediante la anotación:

```
@EnableMethodSecurity(prePostEnabled = true)
```

Esto permite aplicar anotaciones como `@PreAuthorize` para restringir el acceso a los endpoints según los **roles del usuario**, los cuales se validan a través del **token de autenticación**.

```
@PreAuthorize("hasAnyRole('ADMIN', 'SUPERVISOR', 'USER')")
```

Cada método indica explícitamente qué roles tienen permiso de acceso. Por ejemplo

```
@PreAuthorize("hasAnyRole('ADMIN', 'SUPERVISOR', 'USER')")  
@GetMapping("/all")
```

Este endpoint `GET http://localhost:8080/sale/all` solo puede ser accedido por usuarios autenticados que tengan el rol `ADMIN`, `SUPERVISOR` o `USER`.

- **ADMIN y SUPERVISOR:** Tienen acceso completo. Pueden listar, crear, actualizar y eliminar ventas.
- **USER:** Consultar y actualizar una venta.

Este diseño permite que cada operario (**USER**) registre el estado de su trabajo, marcando cuándo una unidad de producción está lista para el siguiente paso, hasta completar el proceso de producción.

Este es el **único controlador accesible** para los usuarios con rol **USER**, lo que garantiza que puedan participar en el flujo de trabajo sin comprometer operaciones administrativas.

Para propagar los cambios entre los distintos usuarios conectados, utilizamos la clase:

```
SimpMessagingTemplate messagingTemplate;
```

que permite **enviar mensajes en tiempo real** a través de **WebSocket**, utilizando el protocolo **STOMP** mencionado anteriormente.

En todos los métodos save, update y delete, después de persistir o eliminar la entidad, se publica el cambio en el canal **"/topic/process"** para que todos los clientes suscritos reciban la actualización automáticamente:

```
messagingTemplate.convertAndSend("/topic/process", processTemplate);
```

El resto de controladores tienen un funcionamiento muy similar, solo se diferencia en las restricciones de cada rol.

8.3. **Plantilla de producto (ProductTemplateController):** Expone los **endpoint** relativos a la gestión de **plantillas** de producto.

Su acceso será restringido a usuarios con el rol administrador y supervisor, ya que estas plantillas definen los flujos de trabajo utilizados en el sistema.

- **Administradores:** pueden **crear, modificar, eliminar y leer** productos.
- **Supervisores:** Pueden **leer** los productos, con el objetivo de detectar errores y evitar problemas en producción, asegurándose que cada producto tenga el proceso correspondiente.

8.4. **Plantilla de proceso (ProcessTemplateController):** Expone los **endpoint** relativos a la gestión de plantillas de **procesos** mediante la anotación:

Su acceso será restringido a usuarios con el rol administrador y supervisor, ya que estas plantillas definen los flujos de trabajo utilizados en el sistema.

- **Administradores:** pueden **crear, modificar, eliminar y leer** procesos.
- **Supervisores:** Pueden **leer** los procesos, con el objetivo de detectar errores o proponer nuevos procesos que se ajusten al marco operativo.

8.5. Plantilla de estados(State Template Controller): Expone los **endpoint** relativos a la gestión de plantillas de **estados** mediante la anotación:

Su acceso será restringido a usuarios con el rol administrador y supervisor, ya que estas plantillas definen los flujos de trabajo utilizados en el sistema.

- **Administradores:** pueden **crear, modificar, eliminar y leer** estados.
- **Supervisores:** Pueden **leer** los estados, con el objetivo de detectar la ausencia de algún estado de producción.

9. Manejo global de excepciones (GlobalExceptionHandler).

Componente centralizado que gestiona las excepciones no controladas lanzadas en los controladores REST de la aplicación. Está anotada con:

```
@RestControllerAdvice
```

Esto le indica a Spring que esta clase interceptará las excepciones de todos los controladores y devolverá una **respuesta HTTP personalizada**, sin necesidad de capturarlas en cada endpoint individualmente.

- 9.1. DataIntegrityViolationException:** Maneja errores relacionados con violaciones a las restricciones de integridad en la base de datos, como duplicados de claves únicas o restricciones de claves foráneas.
- 9.2. RuntimeException:** Captura cualquier otra excepción en tiempo de ejecución como el siguiente:

```
throw new RuntimeException("No sale found with code: " +  
sale.getCode());
```

Producido en SaleService si al actualizar una venta el código no existe.

3.4. Frontend(Cliente) - React.

React es una **biblioteca** de **JavaScript** para construir **interfaces de usuario**. Se utiliza en la creación de aplicaciones web interactivas, rápidas y dinámicas.

La interfaz se construye a partir de **componentes reutilizables**, lo que facilita el **mantenimiento** y **escalado** del código.

Además de gestionar la vista, React puede hacer **llamadas a APIs** para obtener datos JSON y renderizarlos de forma atractiva y eficiente para el usuario

También puede suscribirse a canales STOMP para **recibir actualizaciones** instantáneas desde el backend sin necesidad de hacer peticiones repetitivas. Esto es especialmente útil en una aplicación que requiere actualización en **tiempo real**.

1. Api.

Esta carpeta se compone de ficheros **JavaScript** que contienen **llamadas a la API** de Spring Boot mencionada en el backend.

Todas constan de una constante **BASE_URL** que apunta a la dirección del controlador, por ejemplo a procesos:

```
const BASE_URL = "http://localhost:8080/process";
```

Y seguido las **funciones** para realizar la **petición** , por ejemplo obtener todos los procesos:

```
export const getAllProcess = async () => {
  const token = sessionStorage.getItem('token');

  const res = await fetch(`${BASE_URL}/all`, {
    headers: {
      "Authorization": `Bearer ${token}`
    }
  });
  return res.json();
};
```

1.1. Async: Declara una función como asíncrona y podemos hacer uso de **await** para esperar la resolución de promesas sin bloquear el hilo principal.

1.2. Token: Obtenemos el token del usuario guardado en el **sessionStorage** del navegador, específicamente el token para autenticar peticiones.

- 1.3. Res: Hacemos la petición HTTP usando fetch junto a BASE_URL y el endpoint `/all`, incluye un **header** llamado `"Authorization"` con el valor `"Bearer <token>"`. Esto es un estándar común para enviar tokens JWT en APIs seguras..
- 1.4. Json: extrae el cuerpo de la respuesta HTTP y lo convierte de JSON a un objeto JavaScripty retornamos la promesa con los datos ya parseados.

Esta estructura es muy común para interactuar con APIs protegidas por autenticación. Además de realizar peticiones **GET** podemos utilizar otros métodos HTTP como **POST**, **PUT**, **DELETE**, dependiendo de la operación necesaria.

La información puede viajar en el cuerpo, especificando el tipo de contenido en la cabecera:

```
const res = await fetch(`${BASE_URL}/update`, {  
  method: "PUT",  
  headers: {  
    "Content-Type": "application/json",  
    "Authorization": `Bearer ${token}`  
  },  
  body: JSON.stringify(process),  
});
```

o como parte de la URL:

```
const res = await fetch(`${BASE_URL}/delete/${id}`, {  
  // ...  
});
```

2. WebSocket.

Archivo **JavaScript** que define un **hook personalizado de React**, se conecta a un **WebSocket** usando **STOMP** sobre **SockJS**.

Recibe 3 funciones :

```
const websocket = (onMessage, onDeleteMessage, endPoint) => {
```

- **onMessage**: Función que se llama cuando llega un mensaje nuevo, es decir se ha añadido una entidad.
- **onDeleteMessage**: Función que se llama cuando llega un mensaje de tipo "delete".
- **endPoint**: nombre del canal o ruta del topic como **"sale"** o **"process"**.

A continuación recuperamos el token y establecemos la dirección del socket:

```
const socket = new SockJS(`http://localhost:8080/ws?token=${token}`);
```

Declaramos el cliente que se va a conectar y nos suscribimos a un los **topics**:

- **/topic/\${endPoint}**: Para mensajes normales como los metodos save o update.
- **/topic/\${endPoint}-deleted**: Para mensajes de eliminación producidos en los metodos delete.

Y redirigimos la información a las funciones correspondientes y la trataremos en los componentes de React.

Si hay errores STOMP como un topic no encontrado o token inválido, los muestra en consola.

```
onStompError: (frame) => {
  console.error('Broker reported error: ' +
    frame.headers['message']);
  console.error('Additional details: ' + frame.body);
},
```

Activa la conexión y guarda la referencia al cliente STOMP.

```
client.activate();

clientRef.current = client;
```

Por último si el componente se desmonta, lo desconectamos del WebSocket:

```
return () => {
  clientRef.current.deactivate();
};
```

Este enfoque es ideal para aplicaciones donde se necesitamonitorrear procesos o tareas.

3. Componentes.

Es esta carpeta se encuentra todos los componentes que forman la aplicación y está dividida por:

3.1. Cabecero (header): Representa el header de la web, es decir el menú contiene:

- **Logotipo:** Nombre de la aplicación y si mantenemos el cursor encima, muestra la versión de la aplicación y el nombre del usuario.
- **Botón desplegable:** Despliega un menú para pantallas pequeñas, como las de un móvil o tablets, visible sólo si la pantalla es inferior a 768px.
- **Menú para pantallas grandes:** Habilitado por defecto, se esconde si la pantalla es inferior a 768px.
- **Menú para pantallas pequeñas:** Visible sólo al pulsar el botón desplegable

A parte de controlar el acceso a los endpoint en base al rol, aquí seleccionamos qué opciones de menú será visible para cada rol:

- **Administrador:** Accede a todas las pestañas.
- **Supervisor:** Accede a Ventas, productos, procesos y estados.
- **Usuario:** No ve el menú pues sólo puede acceder a ventas.

3.2. Main.

Body: Contiene las distintas pestañas del menú y son renderizadas en función del valor de la constante **activeSection**.

3.3. Componentes de ventana Ventas.

- **SalesList:** Esta ventana permite visualizar y cargar ventas. Se conecta a un WebSocket mediante:

```
websocket(handleWebSocketUpdate,handleWebSocketDelete, "sale");
```

para recibir actualizaciones en tiempo real.

También permite cargar ventas desde un archivo Excel gracias a la clase **SaleProcessor**.

- **SaleCard: Venta individual** dentro de una lista. Su propósito principal es **mostrar la información de una venta**, incluyendo sus **productos** y **editar o eliminar la venta** si el usuario tiene permisos. Es desplegable, para facilitar la navegación rápida, gracias a **AnimatePresence y motion.div** contamos con una transición suave a la hora de plegar o desplegar el contenido.

- **ProductOnSaleCard:** Representa ProductTemplateDTO **en la venta**, compuesto por componentes, cantidades a producir y unidades de producción asociadas. Es interactivo, expandible y está diseñado para usarse dentro de una venta.
- **ProductionUnitCard:** Representa **una unidad individual de producción** dentro de una venta. Permite editar su número de **carro** y gestionar el avance del **proceso productivo**, para ello hemos de seleccionar el proceso actual. Sigue un código de colores.
 - **Azul:** Proceso actual
 - **Verde:** Proceso finalizado
 - **Rojo:** El proceso fue completado, pero el producto tuvo una incidencia y se ha retrocedido en la producción.
- **SaleModal:** Muestra una **ventana emergente** con la información detallada de una venta. Cuando añadimos una venta se aparece para revisar la información relativa a una venta antes de guardarla en el sistema.

3.4. Componentes de ventana productos.

- **ProductList:** Carga todos los productos de la base de datos, parecido a **SaleList**, nos permite visualizar, crear, editar y eliminar. Integra funciones de carga desde el backend, sincronización en tiempo real mediante WebSocket, y procesamiento de archivos **Excel** para **importar** datos.
- **ProductCard:** Muestra la información de un **producto individual** y permite editarlo o eliminarlo si el usuario tiene rol de administrador. Usa botones para guardar o cancelar los cambios.
- **ProcessOnProductCars:** Representa el **proceso asociado a un producto**, cuenta con un menú desplegable para asignar el proceso específico cuando está en modo edición, si no, muestra la descripción del proceso y una lista de sus estados, cada uno numerado por orden y lo podemos encontrar dentro de **ProductCard**.

3.5. Componentes de ventana procesos.

- **ProcessList:** Muestra una **lista dinámica de procesos** que permite visualizar, agregar, editar y eliminar procesos de producción. También se actualiza en tiempo real mediante **WebSockets**.
- **ProcessCard:** Representa un **proceso de producción**, en modo edición podemos agregar estados desde un menú flotante y cambiar la descripción, siempre que el usuario tenga rol de administrador.

3.6. Componentes de ventana estados.

- **StatesList:** Componente encargado de gestionar y mostrar la lista de estados, mantiene la lista actualizada gracias a los WebSocket. Podemos crear, actualizar o eliminar estados, reflejando estos cambios automáticamente en la interfaz.
- **StateCard:** Representa de forma individual un **estado** dentro de la lista de estados. Muestra la descripción y el identificador del estado, permitiendo a los usuarios con rol de administrador editar o eliminar ese estado. En **modo edición**, el componente muestra un campo de texto para modificar la descripción, junto con un botón para guardar los cambios. Si no está en modo edición, muestra la descripción como texto.

3.7. Componentes de inicio de sesión.

- **Login:** Representa la interfaz y lógica de la **autenticación de usuarios**, consta de 2 inputs, para el nombre de usuario y la contraseña, y hace una llamada a la API. En caso positivo **guarda el token** devuelto por el endpoint y lo guarda como un objeto en la sesión del navegador.
- **UserProvider:** Define un **contexto global de usuario** en React, usado principalmente para gestionar la autenticación del usuario y poder acceder al token **almacenado en la sesión**. Establece si el usuario se ha autenticado, el nombre del usuario, el rol, la sección que debe cargar, restringe el acceso en caso de fallar la autenticación y muestra el error.

3.8. Carpeta Utils.

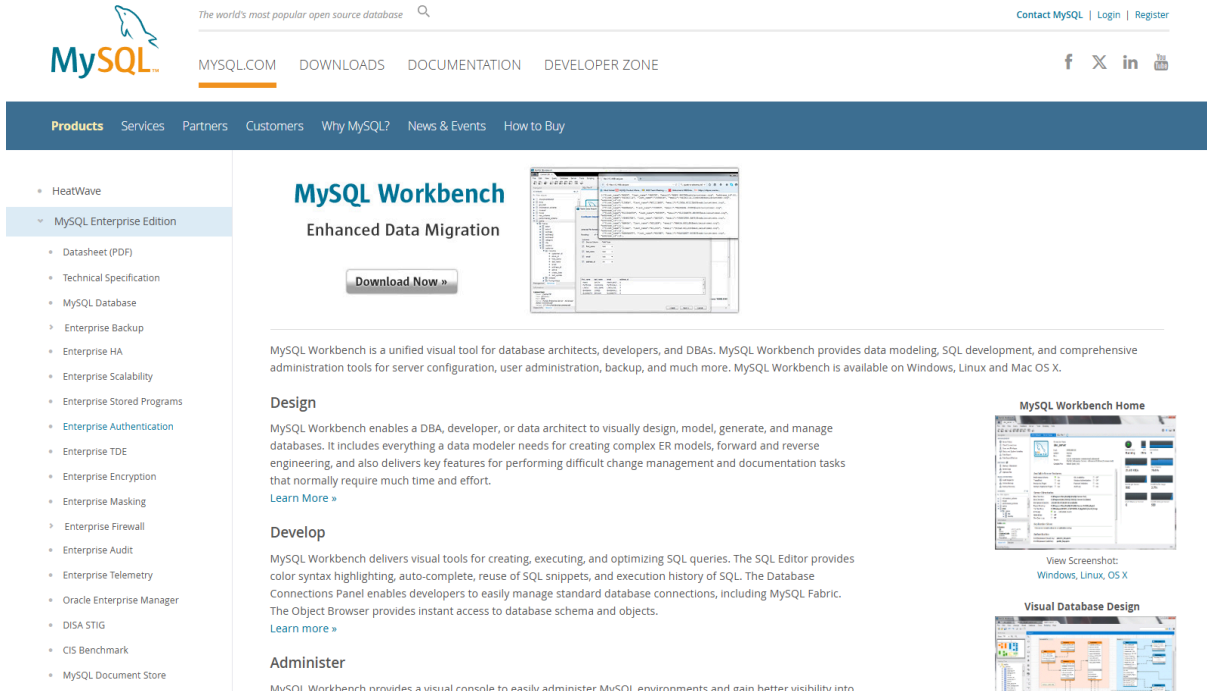
- **ProductProcesor**: Archivo JavaScript encargado de **procesar un documento Excel** sobre productos y transformarlo en una lista de productos. Busca las cabeceras "CÓDIGO", "DESCRIPCION" y "PROCESO" con la clase **FileReader** y obtiene sus **índices** para extraer los datos, por último se mapea a una lista de entidades **ProductTemplate** en formato **JSON**.
- **SaleProcesor**: Archivo JavaScript encargado de **procesar un documento Excel** sobre una venta y transformarlo en una venta. Busca las cabeceras "Código", "Descripción" y "Cantidad" con la clase **FileReader** y obtiene sus índices para extraer los datos, por cada producto también extrae sus componentes, por último devuelve una entidad Sale en formato **JSON**.
- **Utils**: Esta clase tiene un conjunto de funciones:
 - **supressText**: Vacía el texto si coincide con el placeHolder.
 - **handleClick**: Maneja el color de una lista de estado, dependiendo de dónde seleccione el usuario.
 - **setColor**: Devuelve el color de cada estado.

4. Instalación.

Para ejecutar y probar nuestra aplicación solo será necesario el IDE IntelliJ IDEA y una base de datos MYSQL.

4.1. Instalación de entorno MYSQL.

Para instalar mysql de forma sencilla y con interfaz usaremos [MySQL Workbench](#):

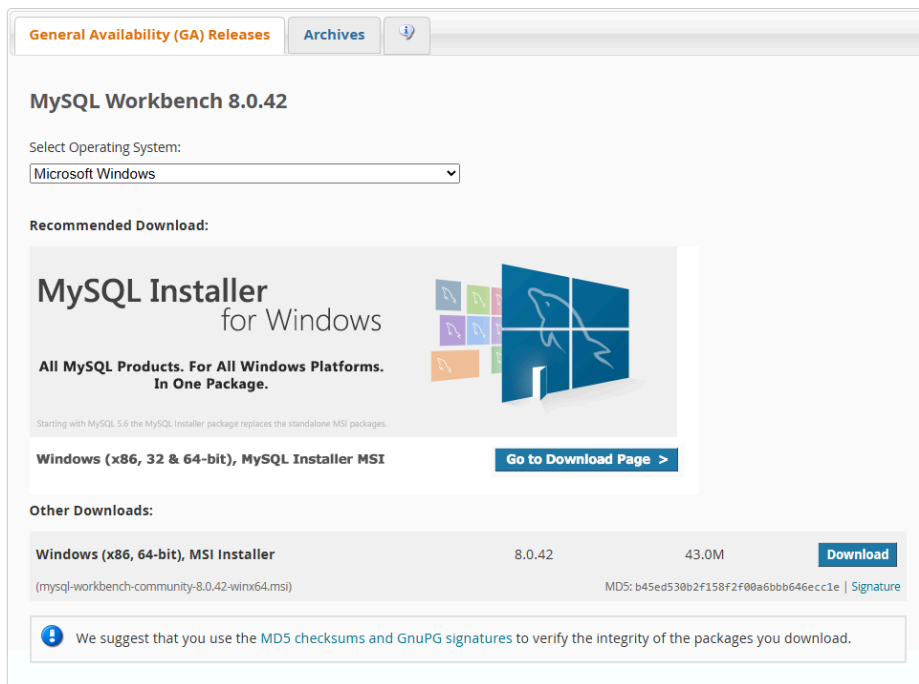


The screenshot shows the MySQL website's MySQL Workbench page. The header includes the MySQL logo, the tagline "The world's most popular open source database", and navigation links for MySQL.COM, DOWNLOADS, DOCUMENTATION, and DEVELOPER ZONE. A sidebar on the left lists various MySQL products and services. The main content area features the MySQL Workbench logo, a "Download Now" button, and a description of the tool. It highlights three main areas: Design (visual design, modeling, and management), Develop (SQL development, execution, and optimization), and Administer (visual console for administration). Two small screenshots on the right show the MySQL Workbench Home screen and a Visual Database Design diagram.

Pulsamos en **download now** y descargamos el instalador de nuestro S.O, en mi caso Windows:

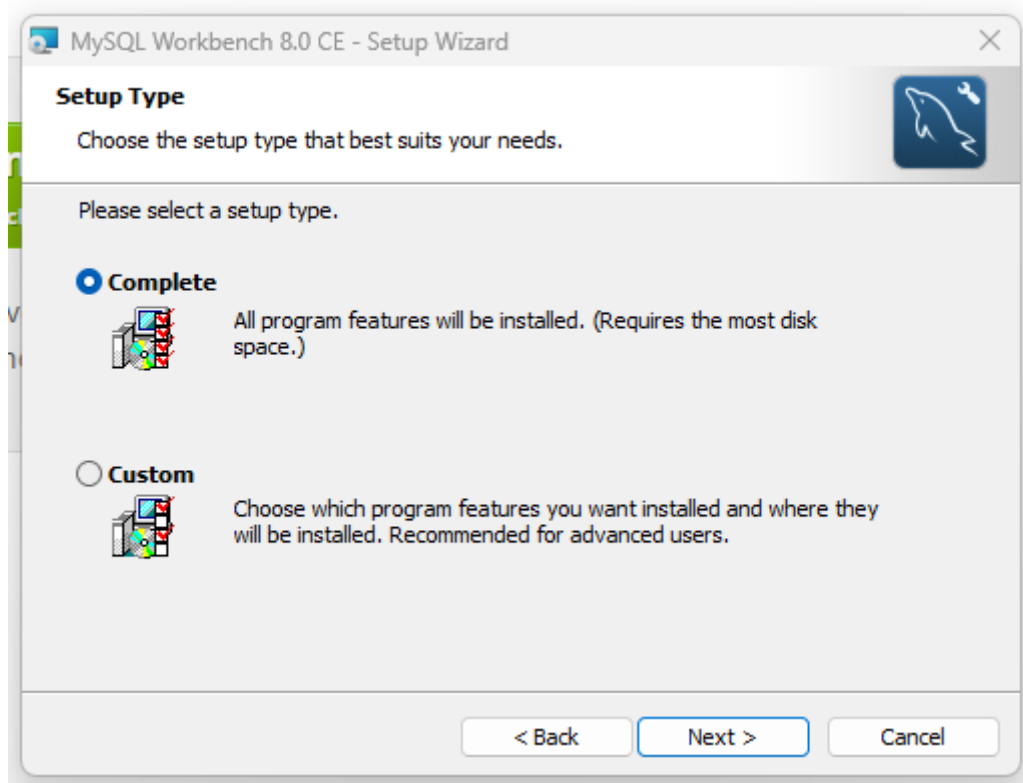
MySQL Community Downloads

MySQL Workbench

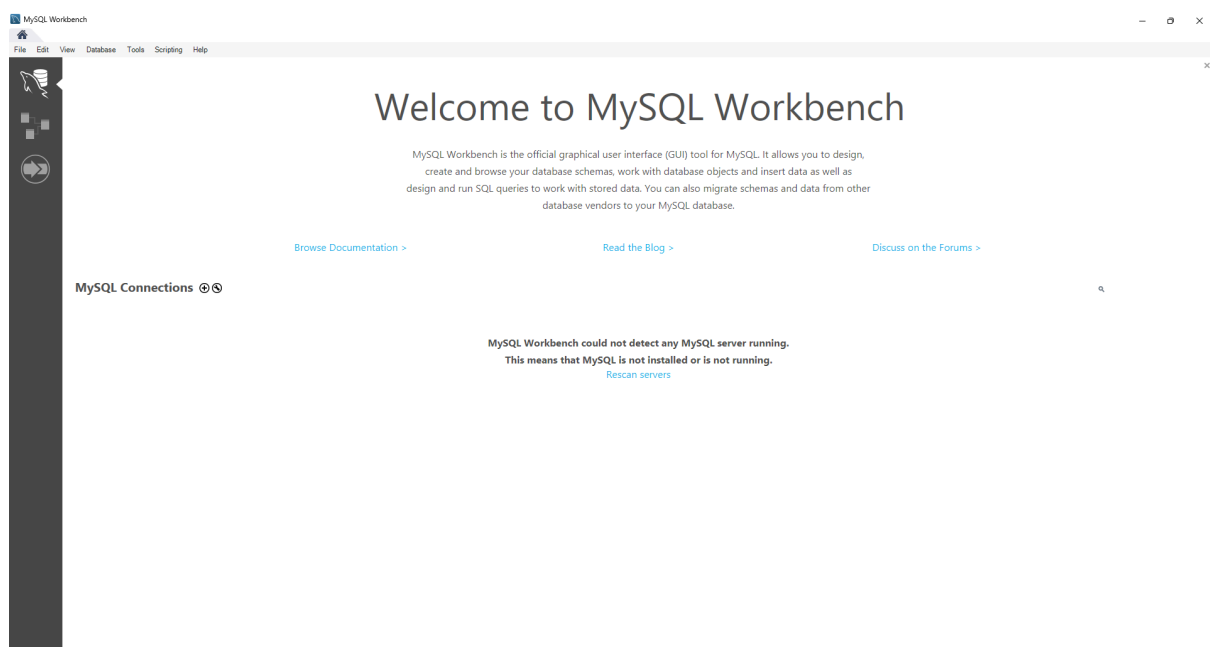


The screenshot shows the MySQL Community Downloads page for MySQL Workbench 8.0.42. The page has tabs for "General Availability (GA) Releases", "Archives", and "Downloads". The "General Availability (GA) Releases" tab is selected. The page displays the MySQL Workbench 8.0.42 download page. It includes a "Select Operating System:" dropdown menu with "Microsoft Windows" selected. Below this, it shows the "Recommended Download:" section with a large image of the "MySQL Installer for Windows" package. The package is described as "All MySQL Products. For All Windows Platforms. In One Package." and includes a "Go to Download Page" button. The "Other Downloads:" section lists the "Windows (x86, 64-bit), MSI Installer" package, version 8.0.42, with a size of 43.0M. It provides the MD5 checksum and a link to the signature. A note at the bottom suggests using MD5 checksums and GnuPG signatures to verify the integrity of the packages.

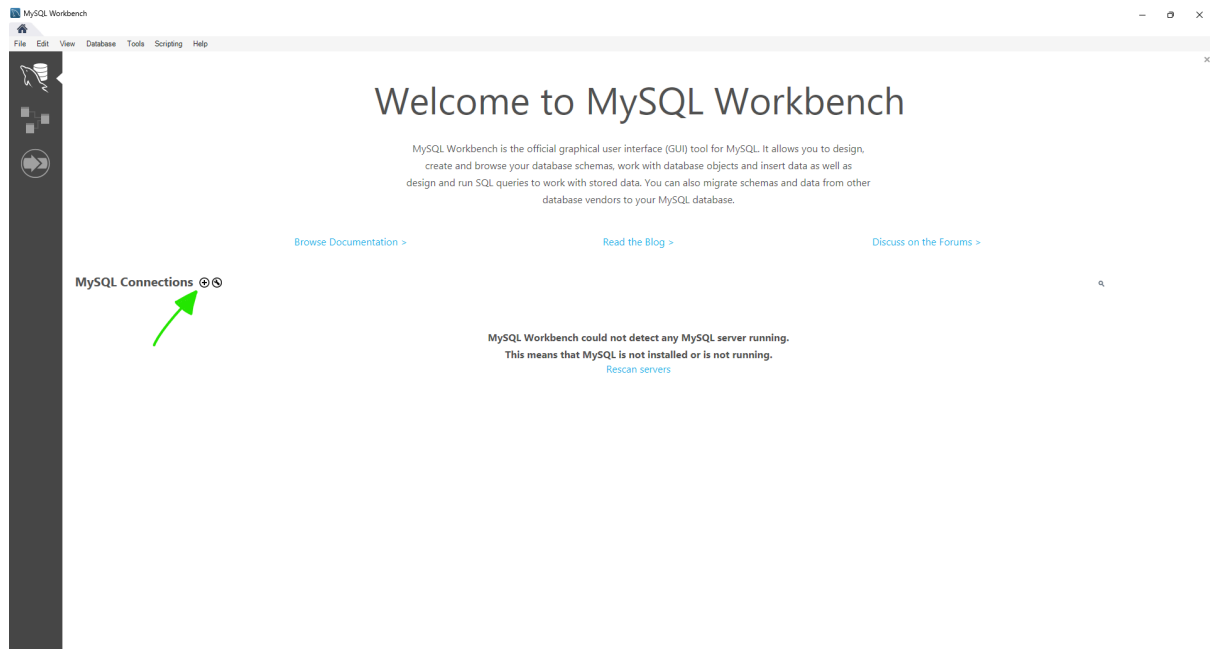
Ejecutamos el instalador y le damos a siguiente hasta esta ventana:



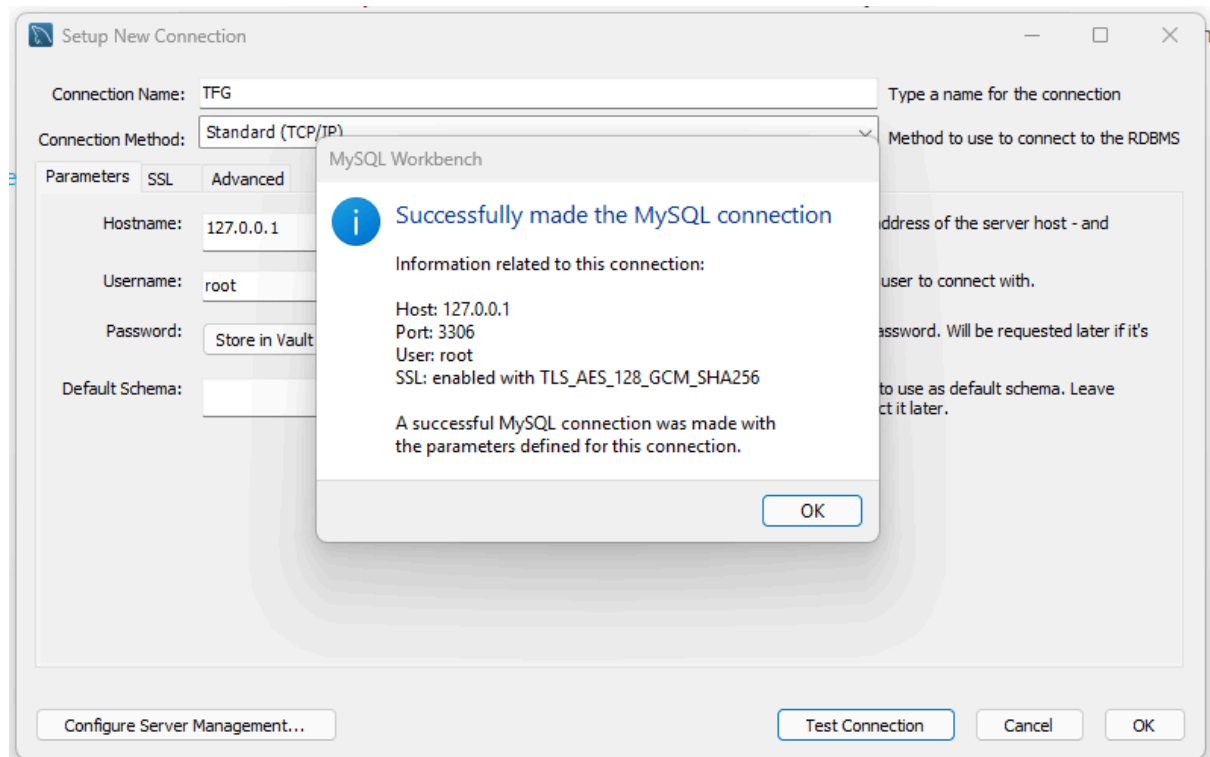
Seleccionamos **Complete** para instalar todas las funciones, pulsamos finalizar y lo abrimos:



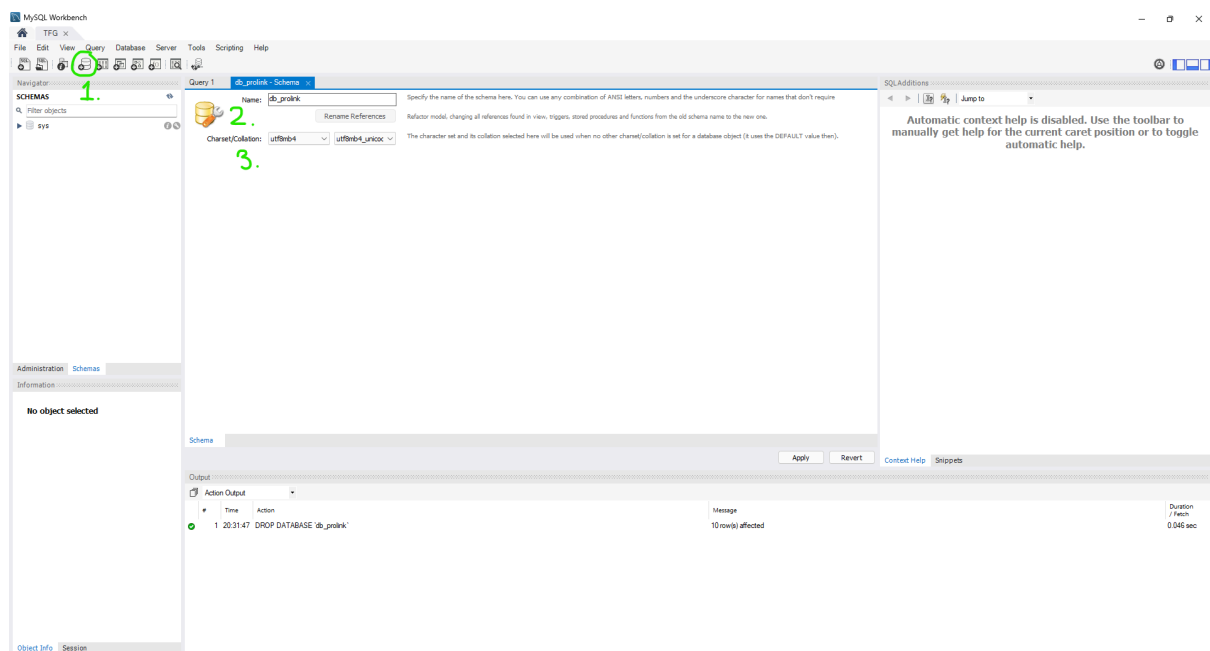
Para añadir una nueva conexión pulsamos al botón más que se aprecia en la parte izquierda MySQL Connections +



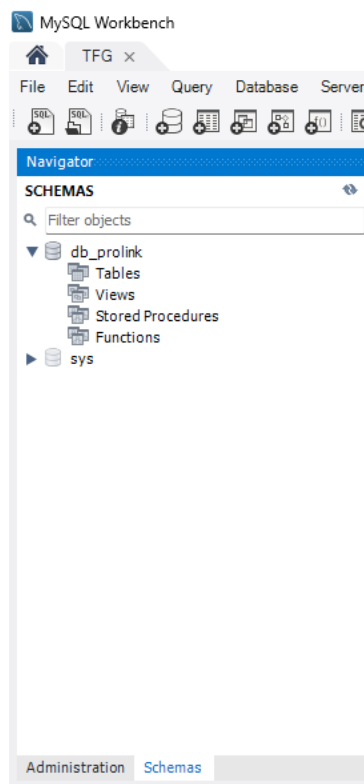
Establecemos el nombre de la conexión, la dirección, en este caso trabajaremos en localhost y probamos la conexión:



Aceptamos y aparecerá nuestra nueva conexión, damos doble click y se abrirá la conexión:



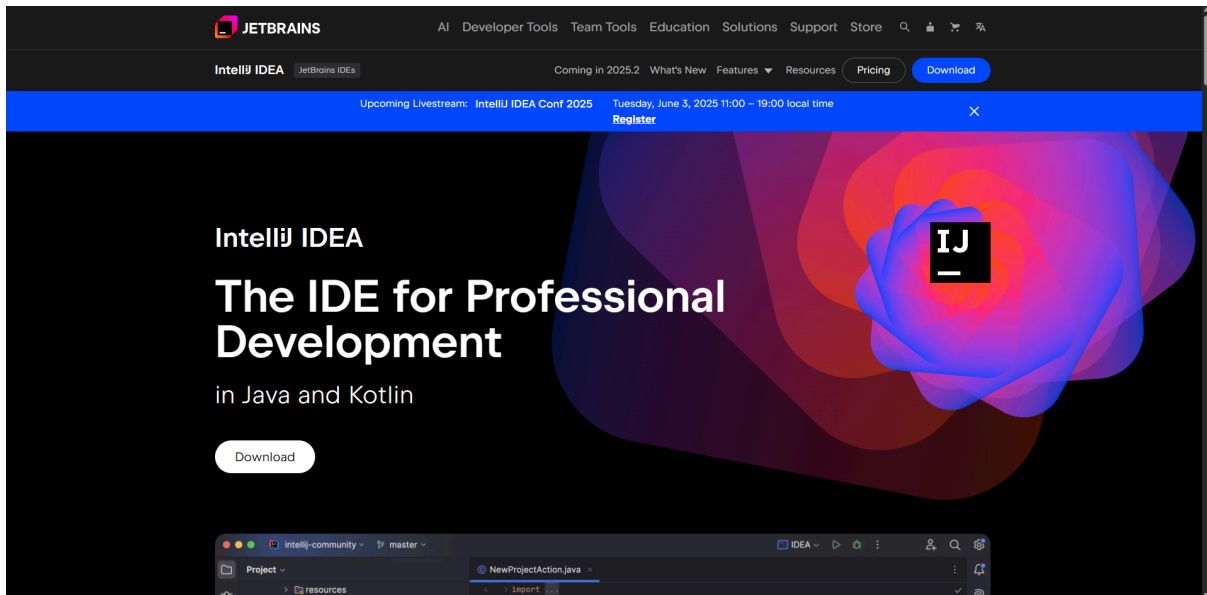
Para añadir una base de datos pulsamos en 1., escribimos el nombre y tipo de charset utf8mb4 y utf8mb4_unicode_520_ci, aplicamos y aceptamos:



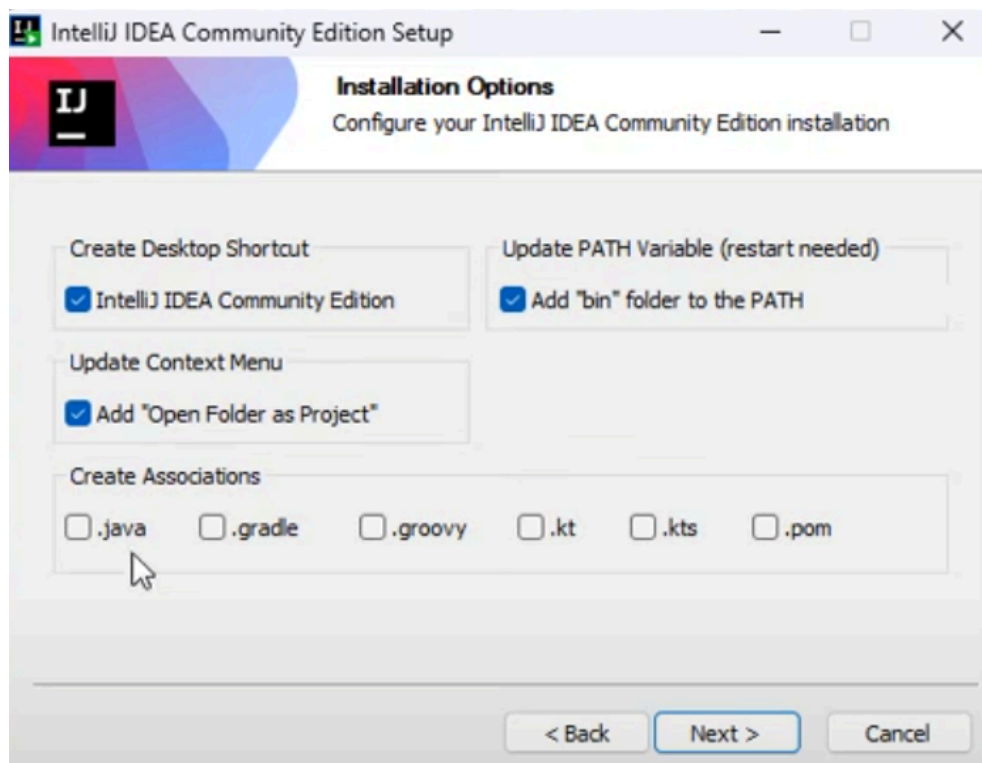
Aparece nuestra base de datos nueva, con esto hemos terminado la instalación de mysql en el entorno Windows y está lista para ser usada por nuestro backend.

4.2. Instalación de entorno IntelliJ IDEA.

Para instalar IntelliJ IDEA accedemos a su [web](#):



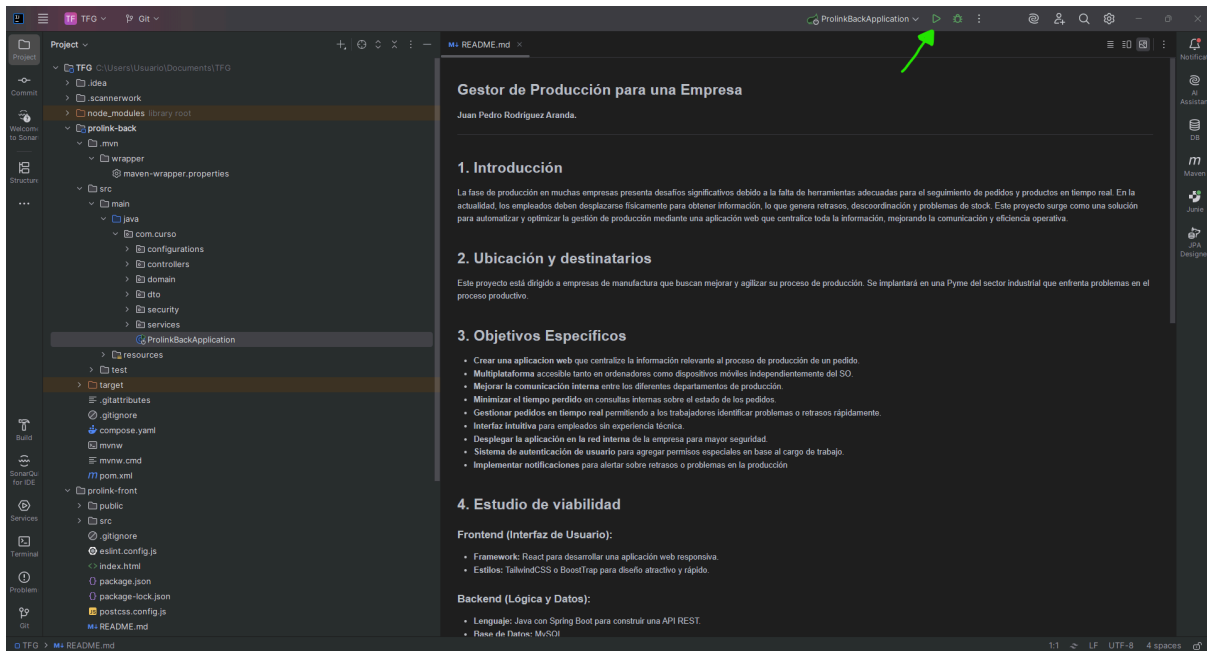
Igual que en Mysql, pulsamos a descargar y seleccionamos el instalador para nuestro SO, pulsamos a siguiente hasta esta pestaña y seleccionamos las siguientes opciones:



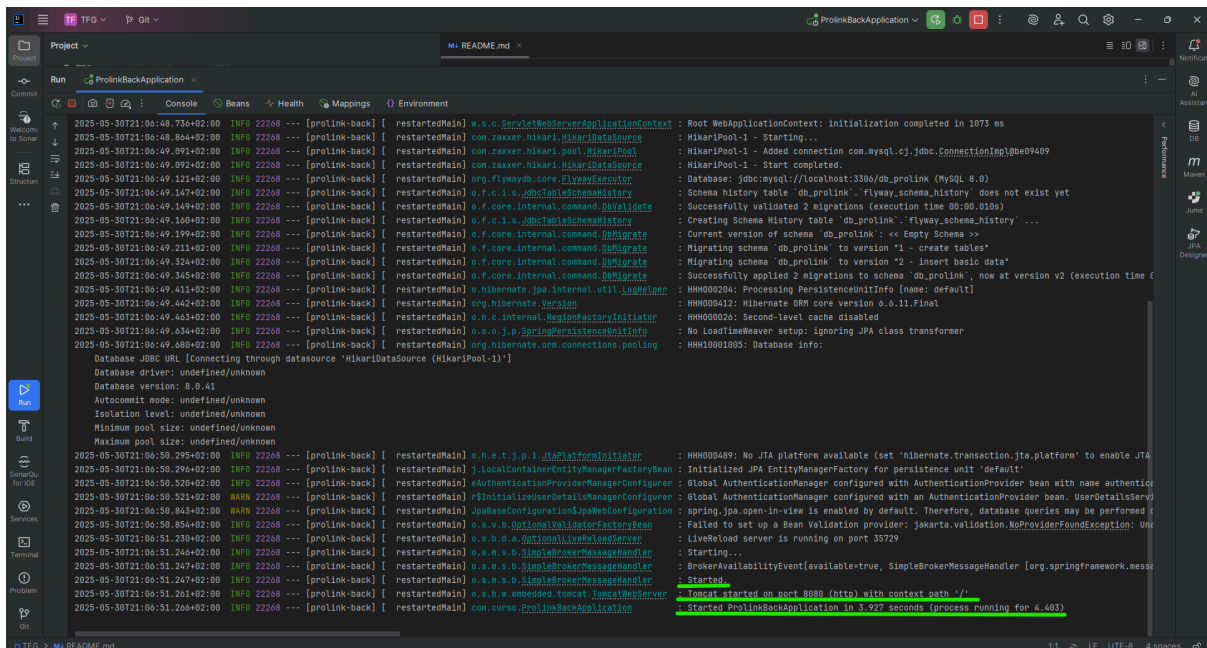
Pulsamos install y ya hemos acabado.

4.3. Ejecutar nuestro servicio API Rest.

Abrimos IntelliJ File -> Open y seleccionamos la carpeta TFG, veremos el anteproyecto:



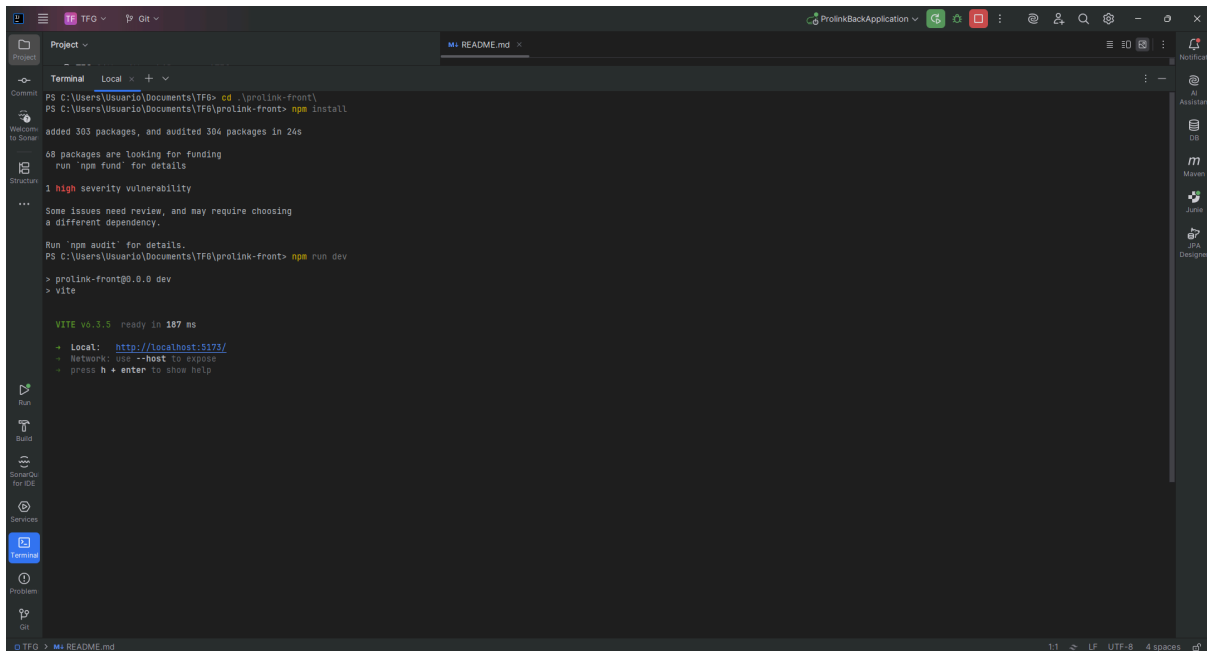
Pulsando el botón run (seleccionado con la flecha) para lanzar nuestra API Rest, si todo funciona correctamente, deberíamos ver este log:



La primera línea subrayada marca que se inició exitosamente el código, las segunda que el servicio está disponible por servidor Tomcat en el puerto 8080 y la tercera que la aplicación está corriendo y el tiempo que ha tardado en ejecutarse.

4.4. Ejecutar nuestra interfaz.

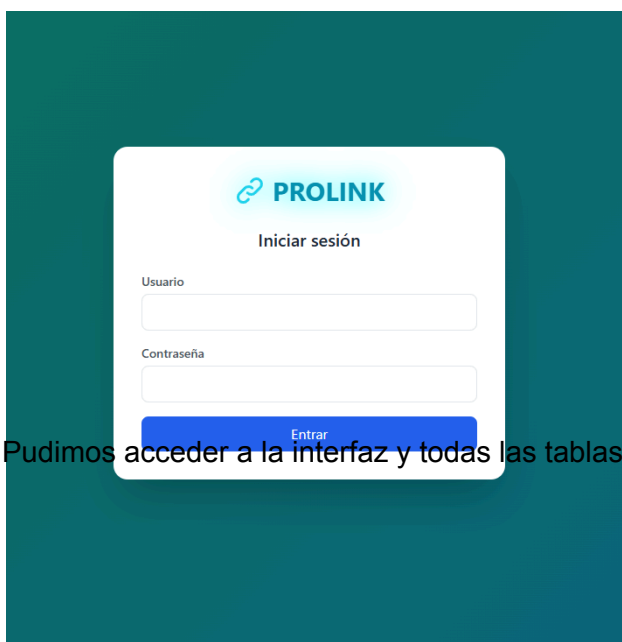
Desde la terminal accedemos a la carpeta prolink-front y ejecutamos los siguientes comandos:



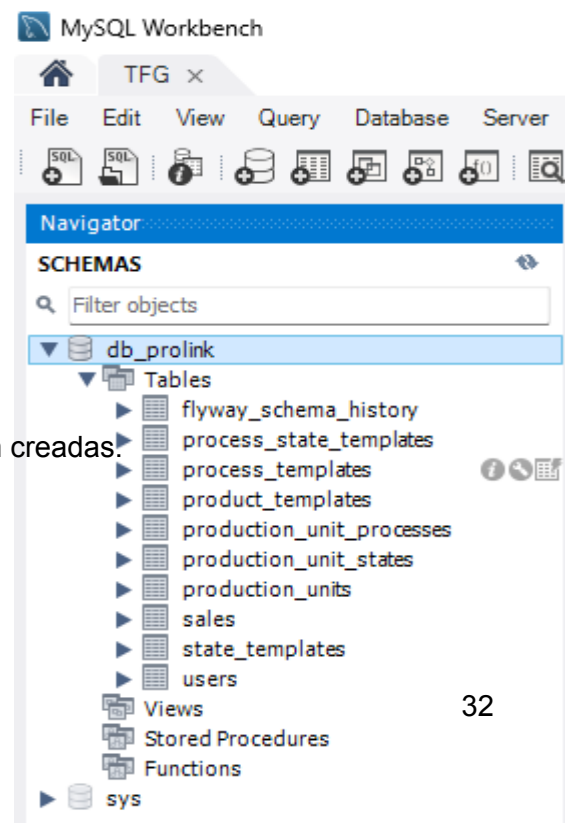
```
PS C:\Users\Usuario\Documents\TFG> cd .\prolink-front\  
PS C:\Users\Usuario\Documents\TFG\prolink-front> npm install  
added 303 packages, and audited 304 packages in 24s  
  
48 packages are looking for funding  
  run `npm fund` for details  
  
1 high severity vulnerability  
Some issues need review, and may require choosing  
a different dependency.  
Run `npm audit` for details.  
PS C:\Users\Usuario\Documents\TFG\prolink-front> npm run dev  
prolink-front@0.0.0 dev  
vite  
  
VITE v6.3.5 ready in 187 ms  
  Local: http://localhost:5173/  
  Network: use --host to expose  
  press h + enter to show help
```

- Npm install: Descarga todas las **dependencias** definidas en nuestro archivo package.json.
- Npm run dev: Ejecutar un **script personalizado** definido en el archivo package.json, es decir iniciamos el front con el perfil dev.

Pulsando control + click izquierdo en el link accedemos a la interfaz:




Pudimos acceder a la interfaz y todas las tablas fueron creadas.



5. Resultado.

Pequeño video demostración:

 Desktop 2025.05.31 - 11.48.29.02.mp4

6. Conclusiones y nuevas propuestas:

Esta aplicación ayudará a las empresas a gestionar y monitorear su producción de forma eficiente, pudiendo llevar un control más preciso, evitando el tiempo perdido entre consultas internas de la empresa

6.1. Ventana de usuarios.

Los administradores tendrán una ventana donde poder crear, modificar y eliminar usuarios.

6.2. Sistema de notificaciones.

Cuando un pedido sufra retrasos, al iniciar sesión, aparecerá una notificación avisando de este problema

6.3. Registro de acciones.

Cada acción quedará registrada en un log, contendrá el nombre de usuario y la acción realizada

6.4. Sistema de búsqueda.

Los usuarios podrán realizar búsquedas específicas ya sea por nombre, fecha, código o id.

7. Referencias bibliográficas y documentales

- [Spring security y autenticación basada en Tokens JWT.](#)
- [React](#)