

INNOVA66

Sistema interno de gestión
técnica y administrativa



Rolando Tomalo Aguilar
2.º Desarrollo de Aplicaciones Multiplataforma
les Ítaca

1. Documento de Descripción del Proyecto	5
1.1. Contexto del proyecto	5
1.1.1. Ámbito y entorno	5
1.1.2. Análisis de la realidad	5
1.1.3. Solución y justificación de la solución propuesta	6
1.1.4. Destinatarios	7
1.2. Objetivo del proyecto	8
1.3. Objetivo del proyecto en lengua extranjera	8
2. Documento de Acuerdo del Proyecto	9
2.1. Requisitos funcionales y no funcionales	9
Requisitos funcionales	9
Requisitos no funcionales	11
2.2. Tareas	12
2.3. Metodología a seguir para la realización del proyecto	12
2.4. Planificación temporal de tareas	13
2.5. Presupuesto (gastos, ingresos, beneficio)	13
Coste por horas de desarrollo	14
Gastos adicionales	14
Presupuesto total estimado: 2.565 – 2.625 €	15
2.6. Contrato/Pliego de condiciones	15
2.7. Análisis de riesgos	17
3. Documento de análisis y diseño	18
3.1. Modelado de datos	18
Diagrama Entidad-Relación	19
3.2. Análisis y diseño del sistema funcional	20
3.3. Análisis y diseño de la interfaz de usuario	21
Guía de Estilo Visual	22
1. Tipografía	22
2. Componentes UI	23
3. Iconografía	24
4. Navegación	24
5. Tema oscuro (opcional futuro)	24
6. Ejemplo	25
3.4. Diseño de la arquitectura de la aplicación	26
3.4.1. Tecnologías/Herramientas usadas y descripción de las mismas	26
3.4.2. Arquitectura de componentes de la aplicación	27
4. Documento de implementación e implantación del sistema	28
4.1 Implementación	28
1. Preparación del backend (Spring Boot)	28
2. Módulo de Autenticación y Usuarios	29
3. Módulo de Jornadas y Actividades	29
4. Módulo de Materiales e Inventario	30
5. Módulo de Clientes, Contactos y Proyectos	30

6. Módulo de Eventos del Calendario y Alertas	30
7. Módulo de Historial de Compras	31
8. Módulo de Albaranes	31
9. Implementación del Frontend (Flutter)	31
10. Control de versiones y pruebas	32
Conclusión de la implementación	32
4.2 Pruebas	33
Metodología utilizada	33
Fases de prueba	33
Resultado general	33
5. Documento de cierre	35
5.1 Documento de instalación y configuración	35
Requisitos previos	35
Instalación del backend	35
Instalación del frontend (Flutter)	36
Configuración adicional (opcional)	37
5.2. Manual de usuario – Innova66	38
A. USUARIO TÉCNICO (App móvil Android)	38
1. Iniciar sesión	38
2. Registrar jornada laboral	39
3. Registrar actividad diaria	40
4. Añadir materiales usados en la actividad	42
5. Eventos del calendario	42
B. USUARIO ADMINISTRADOR (App de escritorio Windows)	43
1. Iniciar sesión	43
2. Panel de administración	44
3. Gestionar materiales (Inventario)	45
4. Registrar compras de material	45
5. Gestionar clientes y contactos	46
6. Gestionar proyectos	46
7. Eventos del calendario	46
8. Generar albaranes	47
9. Albaranes	48
10. Usuarios	48
11. Seguridad y normas	49
5.3 Resultados obtenidos y conclusiones	49
Resultados obtenidos	49
Conclusiones	51
5.4 Diario de bitácora	51
Fase inicial (abril 2025)	51
Fase de desarrollo principal (abril - mayo 2025)	52
Fase de cierre funcional (finales de mayo 2025)	52
Fase final y revisión (junio 2025)	52
6. Bibliografía	53

7. Anexos	55
Anexo I – Modelo Entidad-Relación (MER)	56
1. Entidades y atributos principales	56
Usuario	56
Jornada	57
Actividad	57
ActividadMaterial	57
Material	58
HistorialCompra	58
Cliente	58
Contacto	59
Proyecto	59
EventoCalendario	59
Alerta	60
Albaran	60
LineaAlbaran	60
2. Relaciones clave	61
3. Observaciones del diseño	61
Anexo II – Endpoints de la API REST	63
Módulo: Actividades	63
Módulo: ActividadMaterial	63
Módulo: Albaranes	63
Módulo: Alertas	64
Módulo: Clientes	64
Módulo: Contactos	64
Módulo: Eventos	64
Módulo: Compras	65
Módulo: Jornadas	65
Módulo: Líneas de albarán	66
Módulo: Materiales	66
Módulo: Proyectos	66
Módulo: Usuarios	66
Anexo III – Licencia de uso	68
1. Nombre del proyecto	68
2. Titularidad	68
3. Acceso al código fuente	68
4. Permisos de uso	68
5. Permisos de modificación	68
6. Uso comercial	69
7. Atribución	69
8. Consideraciones adicionales	69
9. Futuras ampliaciones del sistema	69
Anexo IV – Diario de bitácora	71

1. Documento de Descripción del Proyecto

1.1. Contexto del proyecto

1.1.1. Ámbito y entorno

Este proyecto se sitúa dentro del módulo de Proyecto fin de grado del segundo curso del Ciclo Formativo de Grado Superior en Desarrollo de Aplicaciones Multiplataforma (DAM). Su objetivo es aplicar de forma práctica los conocimientos adquiridos a lo largo de la formación, desarrollando una solución software que responda a necesidades reales de una empresa.

La empresa seleccionada para la realización de este proyecto es Indasor 66 S.L., una compañía con sede en Zaragoza que opera desde hace varios años en el sector de las instalaciones eléctricas y energías renovables. Indasor se dedica al diseño, montaje y mantenimiento de instalaciones eléctricas de todo tipo, así como a la automatización de procesos industriales, instalaciones domóticas, climatización y especialmente al desarrollo e implementación de soluciones basadas en energía solar fotovoltaica.

Su actividad se dirige tanto al ámbito industrial como al comercial y particular, lo que le exige un alto grado de organización, trazabilidad y eficiencia en la gestión de materiales, recursos humanos y técnicos, así como en la supervisión de los múltiples proyectos y mantenimientos que gestionan a diario.

El desarrollo de esta aplicación se realizará considerando el entorno y los recursos reales de la empresa, integrándose en su infraestructura actual (Windows Server y red local), y adaptándose a sus flujos de trabajo existentes, con el fin de ofrecer una solución funcional, útil y viable en el corto plazo.

1.1.2. Análisis de la realidad

Actualmente, Indasor enfrenta una serie de desafíos derivados de la falta de digitalización y automatización en sus procesos internos de gestión. Los principales problemas detectados son:

- **Control manual de las actividades diarias:** Los técnicos rellenan sus partes de trabajo en papel al finalizar cada intervención. Esta información luego es traspasada manualmente por el CEO a hojas de cálculo, lo cual no solo consume tiempo, sino que además dificulta el seguimiento detallado de las tareas realizadas, los materiales empleados y las horas trabajadas.
- **Inventario no integrado:** La gestión de materiales y herramientas se realiza mediante un software de facturación que no está conectado a las intervenciones diarias. Por tanto, no existe un sistema que descuente automáticamente el material utilizado, lo que puede generar errores, desabastecimientos inesperados o falta de

trazabilidad.

- **Gestión de proyectos fragmentada:** Las fechas clave relacionadas con inspecciones de instalaciones, revisiones periódicas o mantenimientos preventivos se anotan de forma manual o en Excel, lo que implica un riesgo elevado de omitir plazos importantes que puedan tener consecuencias legales o técnicas.
- **Ausencia de control estadístico o analítico:** No es posible obtener fácilmente informes con el número de horas trabajadas por técnico, por cliente o por período, lo que limita la capacidad de análisis y la toma de decisiones basada en datos.
- **Averías técnicas:** Algunos equipos instalados por Indasor envían SMS al detectar fallos mediante tarjetas SIM. Sin embargo, estos mensajes se reciben en móviles individuales sin un sistema centralizado que los almacene, notifique o gestione de manera eficiente.

1.1.3. Solución y justificación de la solución propuesta

La solución propuesta consiste en el desarrollo de una aplicación de gestión interna multiplataforma, compuesta por dos entornos: una aplicación de escritorio para ser usada desde la oficina (principalmente por el CEO/administrador), y una aplicación móvil para uso del equipo técnico durante sus intervenciones diarias.

Las principales funcionalidades de la solución propuesta son:

- **Registro diario de actividades:** Cada técnico podrá registrar, desde su dispositivo, los datos de cada intervención en tiempo real o al finalizar la jornada: lugar, cliente, fecha y hora de entrada y salida, tareas realizadas, materiales empleados y observaciones. Esto facilitará la generación automática de albaranes y la trazabilidad completa de cada trabajo.
- **Gestión de inventario:** Se desarrollará un sistema CRUD completo (crear, leer, actualizar, eliminar) para registrar materiales, cantidad, ubicación, proveedor y nivel de stock mínimo configurable. Además, el sistema podrá generar alertas cuando el inventario esté por debajo de un umbral establecido.
- **Gestión de proyectos:** Se integrará un calendario con fechas clave para el cumplimiento de revisiones, inspecciones o legalizaciones. Estas fechas estarán disponibles para el administrador y se podrán sincronizar con dispositivos móviles.
- **Gestión de alarmas por avería (opcional, si el tiempo lo permite):** Se integrará una API como Twilio para centralizar los mensajes SMS enviados por los dispositivos instalados en campo. Esto permitirá crear un historial de alarmas y recibir notificaciones en tiempo real para actuar rápidamente.

- **Informes y análisis:** Se podrá consultar la información registrada por técnicos de forma diaria, semanal, mensual o anual. Estos informes estarán disponibles para cada trabajador o agrupados por cliente o proyecto.

Esta solución proporcionará una mejora radical en la organización interna, la eficiencia operativa y la trazabilidad de las intervenciones. Se digitaliza el flujo de trabajo actual sin cambiar la estructura de la empresa, integrando las herramientas necesarias en su día a día.

1.1.4. Destinatarios

El sistema está destinado a dos perfiles principales dentro de la organización:

- **Administrador (CEO):**
 - Acceso completo al sistema.
 - Visualización y gestión del inventario, calendario de proyectos, registros de actividades y generación de informes.
 - Recepción de alertas importantes (stock bajo, fechas clave, averías).
 - Aprobación o revisión de intervenciones y albaranes generados con el registro diario de los técnicos.
- **Técnicos:**
 - Acceso desde la aplicación móvil únicamente a las funcionalidades necesarias para registrar sus actividades diarias.
 - Sin acceso al inventario, historial o informes.
 - Posibilidad de registrar intervenciones desde distintos puntos en el día, según cambien de ubicación.

El sistema también está planteado para ser escalable, permitiendo en un futuro la inclusión de nuevos roles como supervisores u operarios especializados.

1.2. Objetivo del proyecto

El objetivo general del proyecto es desarrollar una aplicación de gestión interna multiplataforma para la empresa Indasor 66 S.L. que mejore, digitalice y optimice los procesos de control de actividades, gestión de inventario, planificación de proyectos e historial de averías informadas por SMS.

Este objetivo general se desglosa en los siguientes objetivos específicos:

1. **Control diario de actividades:** Digitalizar el registro de las intervenciones de los técnicos, asociándolas automáticamente al cliente y proyecto correspondiente, y permitiendo generar informes automatizados en distintos rangos de tiempo.
2. **Automatización del inventario:** Integrar la gestión de stock con las intervenciones realizadas, para restar automáticamente el material utilizado e informar al administrador cuando sea necesario reponer material.
3. **Gestión de proyectos y tareas pendientes:** Incorporar un calendario que contenga todas las fechas importantes del negocio, como inspecciones, mantenimientos o revisiones técnicas, y generar alertas automáticas.
4. **(Opcional) Gestión de alarmas de averías:** Centralizar en una única interfaz los SMS enviados por los equipos en campo para generar alertas en tiempo real y obtener un historial de los mismos.
5. **Desarrollo multiplataforma:** Asegurar que la solución sea accesible desde dispositivos móviles (Android/iOS) y escritorio (Windows), adaptándose a las rutinas de trabajo en oficina y en campo.
6. **Mínimo coste de infraestructura:** Diseñar una solución viable, económica y sostenible, que pueda funcionar en un servidor local con acceso remoto para facilitar su adopción en la empresa sin necesidad de grandes inversiones iniciales.

1.3. Objetivo del proyecto en lengua extranjera

The general objective of the project is to develop a multiplatform internal management application for the company Indasor 66 S.L. that improves, digitalizes, and optimizes the processes of activity control, inventory management, project planning, and history of faults reported via SMS.

This general objective is broken down into the following specific objectives:

1. **Daily activity control:** Digitize the recording of technicians' interventions, automatically associating them with the corresponding client and project, and allowing the generation of automated reports over different time ranges.
2. **Inventory automation:** Integrate stock management with the interventions carried out, automatically subtracting the used materials and informing the administrator when it is necessary to replenish supplies.

3. **Project and pending task management:** Incorporate a calendar that contains all the important business dates, such as inspections, maintenance, or technical reviews, and generate automatic alerts.
4. **(Optional) Fault alarm management:** Centralize the SMS sent by field teams into a single interface to generate real-time alerts and obtain a history of them.
5. **Multiplatform development:** Ensure that the solution is accessible from mobile devices (Android/iOS) and desktop (Windows), adapting to both office and fieldwork routines.
6. **Minimal infrastructure cost:** Design a viable, economical, and sustainable solution that can run on a local server with remote access to facilitate its adoption within the company without the need for large initial investments.

2. Documento de Acuerdo del Proyecto

2.1. Requisitos funcionales y no funcionales

Requisitos funcionales

Son las funcionalidades que el sistema debe cumplir para satisfacer las necesidades del usuario. Se agrupan según los módulos principales del sistema:

Módulo 1: Control diario de actividades

- RF1.1. El sistema permitirá a los técnicos registrar intervenciones diarias indicando cliente, ubicación, tareas realizadas, materiales empleados, observaciones, y horas de entrada y salida.
- RF1.2. El sistema asociará automáticamente cada actividad a un proyecto mediante un identificador de cliente.
- RF1.3. El administrador podrá consultar informes diarios, semanales, mensuales o anuales por técnico o proyecto.
- RF1.4. El sistema generará automáticamente un albarán con los datos registrados.
- RF1.5. El sistema validará que no se dupliquen jornadas para un técnico en una misma fecha, cumpliendo la normativa vigente en materia de registro laboral.
- RF1.6. El sistema permitirá registrar actividades urgentes sin necesidad de una jornada activa, con el fin de cubrir intervenciones por averías.

Módulo 2: Gestión de inventario

- RF2.1. El administrador podrá registrar, modificar y eliminar artículos del inventario.
- RF2.2. El sistema actualizará automáticamente el stock al registrar una intervención que utilice material.
- RF2.3. El sistema generará alertas cuando un artículo alcance su umbral mínimo definido.
- RF2.4. El administrador podrá establecer niveles de prioridad para cada artículo.
- RF2.5. El sistema permitirá registrar compras con historial detallado, incluyendo proveedor, cantidad, precio unitario, descuento y precio final.
- RF2.6. El sistema permitirá consultar el historial de compras por material o fechas.

Módulo 3: Gestión de proyectos y calendario

- RF3.1. El administrador podrá añadir y editar fechas clave asociadas a cada proyecto (inspecciones, revisiones, mantenimientos, etc.).
- RF3.2. El sistema enviará alertas automáticas al administrador antes de las fechas límite.
- RF3.3. El calendario podrá sincronizarse con dispositivos móviles.
- RF3.4. El calendario permitirá marcar eventos como “pendientes”, “realizados” o “cancelados”, mostrando en pantalla únicamente los pendientes según el tipo de evento.
- RF3.5. Los eventos se clasificarán en cuatro tipos: ITV, Mantenimiento, Revisiones, Seguros, con sus respectivos preavisos (30 o 60 días según el caso).

Módulo 4 (opcional): Centralización de alarmas de averías

- RF4.1. El sistema recibirá automáticamente los SMS enviados por los dispositivos instalados en campo.
- RF4.2. El sistema almacenará un historial de averías y enviará notificaciones al administrador y a los técnicos asignados.

Gestión de usuarios y permisos

- RF5.1. Solo el administrador podrá acceder al inventario, calendario y datos históricos.

- RF5.2. Cada técnico solo podrá registrar actividades, sin acceso a los datos de otros usuarios o historial.

Módulo adicional: Generación y gestión de albaranes

- RF6.1. El sistema generará albaranes agrupando actividades por día, cliente y proyecto.
- RF6.2. El administrador podrá añadir líneas de mano de obra y valorar el albarán (precios y descuentos).
- RF6.3. El sistema mostrará historial de precios por material como ayuda visual durante la valoración.
- RF6.4. El sistema permitirá generar albaranes valorados o sin valorar, y descargarlos en PDF con diseño profesional.

Requisitos no funcionales

- RNF1. La aplicación debe estar disponible en español.
- RNF2. La solución debe ejecutarse en entornos Windows (escritorio) y Android/iOS (móvil).
- RNF3. La base de datos principal será MySQL, alojada en un servidor local Windows Server.
- RNF4. La aplicación debe ser accesible remotamente desde los dispositivos móviles de los técnicos.
- RNF5. El sistema debe permitir una escalabilidad mínima para incluir nuevos módulos o funcionalidades en el futuro.
- RNF6. La interfaz debe ser intuitiva y de fácil uso para personal no técnico.
- RNF7. El sistema debe garantizar la integridad de los datos y su recuperación ante caídas o errores (copias de seguridad).

2.2. Tareas

Aquí se detallan las tareas organizadas por bloques de trabajo:

Nº	Tarea	Descripción
T1	Análisis de requisitos	Revisión de necesidades y planificación con el cliente
T2	Diseño de la base de datos	Modelo entidad-relación y normalización
T3	Diseño de interfaz (UX/UI)	Wireframes y mockups para escritorio y móvil
T4	Configuración del servidor	Preparar Windows Server, MySQL, conexión remota
T5	Desarrollo módulo 1	Registro de actividades e informes
T6	Desarrollo módulo 2	Inventario y alertas de stock bajo
T7	Desarrollo módulo 3	Calendario de proyectos y recordatorios
T8	Desarrollo módulo 4 (opcional)	Centralización de SMS de averías
T9	Integración app móvil	Registro de actividad desde dispositivos
T10	Pruebas funcionales	Verificación de cada módulo con el cliente
T11	Documentación técnica	Manuales de uso, instalación y mantenimiento
T12	Redacción memoria TFG	Documentación oficial del proyecto

2.3. Metodología a seguir para la realización del proyecto

Se seguirá una metodología incremental combinada con elementos de desarrollo ágil, adaptándose a las limitaciones de tiempo y recursos del proyecto.

El desarrollo se dividirá en iteraciones funcionales (entregas parciales) que permitirán validar progresivamente los módulos clave con el cliente:

- Se prioriza el desarrollo de módulos funcionales y básicos.
- Cada iteración incluye desarrollo, pruebas unitarias y validación.

- La comunicación con el CEO será constante para asegurar que los requerimientos se implementan correctamente.
- Se realizarán pruebas prácticas sobre los datos reales de la empresa.

Además, se mantendrá un control de versiones del código (GitHub) y un calendario de entregas parciales para facilitar el seguimiento del avance.

2.4. Planificación temporal de tareas



Nota: Este cronograma puede sufrir ajustes dependiendo del ritmo de desarrollo y el tiempo disponible durante las prácticas.

2.5. Presupuesto (gastos, ingresos, beneficio)

Con el fin de estimar el coste real del desarrollo de este proyecto, se plantea una simulación como si fuera llevado a cabo por un desarrollador autónomo profesional. Se detalla a continuación el presupuesto, calculado sobre una tarifa estándar de 15 €/hora.

Coste por horas de desarrollo

Fase	Tarea	Horas estimadas	Precio/hora (€)	Subtotal (€)
Análisis	Reunión con cliente, planificación	12 h	15 €	180 €
Diseño	Base de datos, UX/UI, servidor	20 h	15 €	300 €
Desarrollo Módulo 1	Registro de actividades e informes	30 h	15 €	450 €
Desarrollo Módulo 2	Inventario y alertas	24 h	15 €	360 €
Desarrollo Módulo 3	Calendario y recordatorios	24 h	15 €	360 €
Desarrollo Módulo 4 (opcional)	Integración SMS y alertas	16 h	15 €	240 €
App móvil	Registro desde Android	20 h	15 €	300 €
Pruebas	Verificación con cliente	10 h	15 €	150 €
Documentación	Técnica y memoria TFG	15 h	15 €	225 €
Total desarrollo		171 h		2.565 €

Gastos adicionales

Elemento	Coste estimado	Observaciones
API de SMS (Twilio u otra)	20–60 € aprox.	Solo si se desarrolla el módulo de alarmas
Servidor local y MySQL	0 €	Ya disponible en la empresa

Software de desarrollo	0 €	Software libre (Java, MySQL, Flutter)
Firebase (notificaciones)	0 €	Plan gratuito suficiente
Total estimado adicional	20–60 €	Dependiendo del uso de la API de SMS

Presupuesto total estimado: 2.565 – 2.625 €

2.6. Contrato/Pliego de condiciones

Contrato de desarrollo de software a medida

Entre:

Indasor 66 S.L., con sede en Zaragoza, en adelante "*el Cliente*",

y

Rolando Tomalo Aguilar, en adelante "*el Desarrollador*",

ACUERDAN:

Objeto del contrato:

El Desarrollador se compromete a realizar el desarrollo de una aplicación de gestión interna multiplataforma para el Cliente, destinada a optimizar la supervisión de actividades técnicas, la gestión de inventario, la planificación de proyectos y, opcionalmente, la centralización de alertas por avería vía SMS.

Alcance y entregables:

- Aplicación de escritorio (Windows) y móvil (Android/iOS)
- Base de datos relacional en MySQL
- Documentación técnica y manual de usuario
- Asistencia durante la instalación y puesta en marcha
- (Opcional) Módulo de alarmas vía API SMS

Plazos de entrega:

- Inicio del proyecto: 17 Marzo de 2025

- Entrega final estimada: 15 de Junio de 2025
- Se contemplan entregas parciales e iteraciones para validación.

Condiciones económicas:

- Importe total: 2.565 € (IVA no incluido)
- Modalidad de pago:
 - 30 % al inicio del proyecto
 - 40 % tras entrega intermedia funcional
 - 30 % tras la entrega final y validación completa
- Posibilidad de facturación por hitos.

Propiedad intelectual:

El código fuente del proyecto desarrollado será propiedad del Desarrollador, quien lo conservará íntegramente. No obstante, se entregará al Cliente una versión compilada y completamente funcional de la aplicación, junto con la documentación técnica necesaria para su uso, instalación y mantenimiento. El Cliente podrá hacer uso libre del sistema dentro del ámbito de su empresa, sin acceso al código fuente, salvo acuerdo posterior expreso entre ambas partes.

Confidencialidad:

Ambas partes se comprometen a mantener la confidencialidad sobre cualquier información técnica o empresarial intercambiada durante el desarrollo.

Soporte y mantenimiento:

Este contrato no contempla soporte técnico posterior a la entrega. No obstante, podrá establecerse un contrato adicional de mantenimiento si el Cliente lo requiere tras la puesta en marcha.

2.7. Análisis de riesgos

Durante el desarrollo del proyecto se han identificado diversos riesgos potenciales que podrían afectar al cumplimiento de los plazos, la calidad del software o la viabilidad técnica de la solución. A continuación, se describen los principales riesgos detectados, su probabilidad de ocurrencia, el impacto que podrían tener en el proyecto y las medidas adoptadas para su mitigación.

Uno de los riesgos más significativos es el retraso por falta de tiempo, con una probabilidad alta y un impacto igualmente alto, dado que el proyecto debe entregarse en un plazo definido. Para mitigarlo, se ha optado por una planificación semanal realista y el uso de una metodología de desarrollo incremental, priorizando siempre los módulos esenciales.

Otro riesgo potencial está relacionado con los cambios en los requisitos por parte del cliente, con una probabilidad media e impacto medio. Para minimizar este riesgo, se han establecido reuniones frecuentes con el CEO de la empresa para validar avances y asegurar que los requerimientos funcionales se están implementando de forma adecuada.

En cuanto a aspectos técnicos, se considera probable la aparición de problemas con el servidor local, con un impacto alto debido a su papel central en la infraestructura. Ante esta posibilidad, se contempla el uso de un entorno de pruebas local alternativo, que permita continuar con el desarrollo en caso de que surjan fallos o incompatibilidades en el entorno real.

También se ha identificado como riesgo la falta de conocimiento previo en el uso de APIs externas, como Twilio, para la integración del módulo de centralización de SMS. Este riesgo presenta una probabilidad alta, aunque su impacto es bajo, ya que se ha considerado una funcionalidad opcional que solo se implementará si el calendario lo permite.

Otro aspecto técnico importante es la posible dificultad en la configuración de la conexión remota entre los dispositivos móviles y el servidor, con una probabilidad media e impacto alto. Para anticiparse a este problema, se realizarán pruebas tempranas de conexión remota y se elaborará una guía de configuración bien documentada.

Por último, la inexperiencia en tareas de despliegue y puesta en marcha de la aplicación supone un riesgo con probabilidad alta e impacto medio. Este riesgo se abordará mediante el estudio de documentación oficial, recursos formativos y el apoyo en comunidades de desarrolladores y foros especializados.

Gracias a esta identificación y planificación anticipada de riesgos, se pretende garantizar la finalización exitosa del proyecto dentro del plazo y con un nivel de calidad adecuado.

3. Documento de análisis y diseño

3.1. Modelado de datos

El sistema Innova66 utiliza un modelo de datos relacional implementado en MySQL, diseñado para representar con precisión los procesos internos de la empresa Indasor 66 S.L. Se partió de un modelo entidad-relación (MER) y se aplicó una normalización hasta tercera forma normal (3FN) para evitar redundancias, garantizar la integridad referencial y facilitar la trazabilidad completa del flujo operativo.

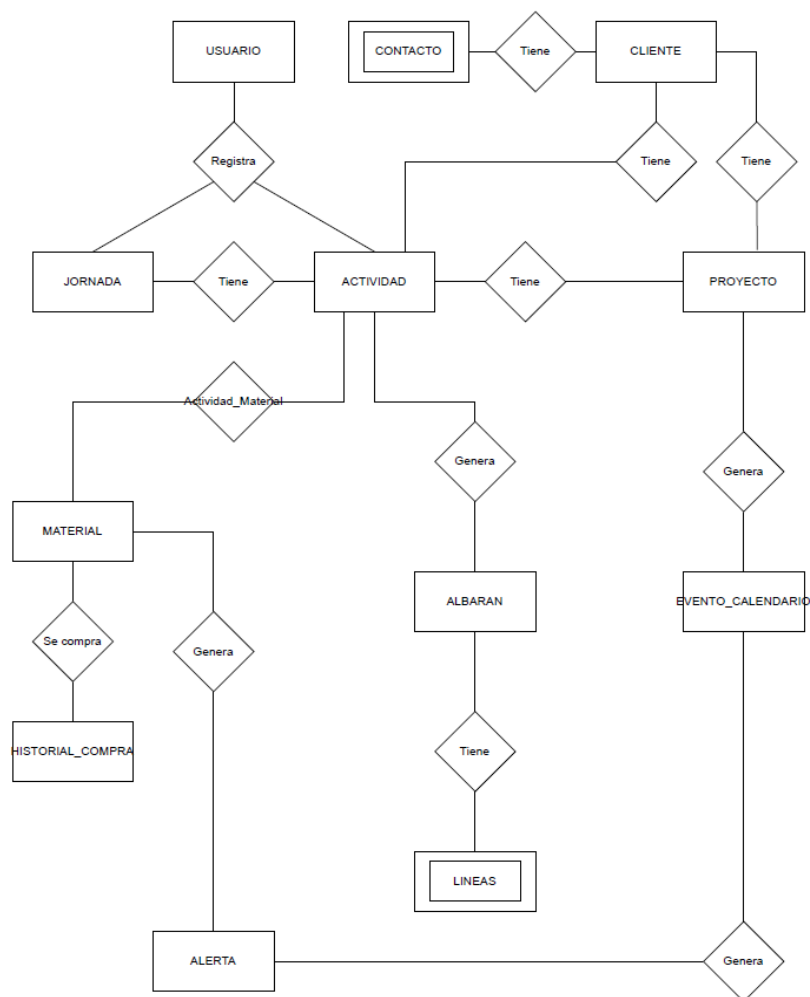
Las entidades clave definidas y sus relaciones permiten cubrir las funcionalidades actuales y futuras de la aplicación. A continuación se detallan:

- **Usuarios:** Representan al personal técnico y al administrador. Cada usuario puede registrar jornadas y actividades, y tiene asignado un rol y datos como teléfono u oficialía.
- **Clientes y Contactos:** Contienen la información de las empresas o personas físicas a quienes se presta servicio. Cada cliente puede tener múltiples personas de contacto asociadas, con sus respectivos cargos, teléfonos y correos.
- **Proyectos:** Corresponden a instalaciones, mantenimientos o intervenciones con seguimiento. Están vinculados a un cliente y pueden tener eventos asociados (como inspecciones o revisiones).
- **Jornadas:** Registro obligatorio diario por técnico. Incluye hora de inicio, fin, si hubo pausa para comer y observaciones generales. Es la unidad base para controlar la actividad laboral.
- **Actividades:** Intervenciones realizadas por los técnicos, con o sin jornada previa. Recogen la fecha, horario, tareas realizadas, materiales utilizados y observaciones. Están relacionadas con jornada (opcional), usuario, cliente y proyecto.
- **Materiales:** Elementos del inventario clasificados por tipo y unidad de medida. Incluyen cantidad en stock, stock mínimo, y se actualizan automáticamente al registrar actividades o compras.
- **Historial de Compras:** Registra las compras realizadas por proveedor, fecha, cantidad, precio unitario, descuento aplicado y precio final. Incrementa el stock automáticamente al registrar una compra.
- **ActividadMaterial:** Tabla intermedia para registrar cuántas unidades de cada material se usaron en una actividad.
- **Albaranes:** Documentos generados a partir de una o varias actividades. Pueden ser sin valorar (solo cantidades) o valorados (con precios, descuentos y totales).

Incluyen línea de mano de obra agregada por el administrador.

- **Líneas de Albarán:** Cada línea puede representar materiales o mano de obra. Permiten edición manual por parte del administrador y cálculo del total del albarán.
- **Alertas:** Notificaciones relacionadas con stock bajo de materiales o fechas clave de eventos próximos. Tienen un estado (activa, resuelta) y se generan o cierran automáticamente según la lógica de negocio.
- **Eventos del Calendario:** Fechas importantes clasificadas en tipos (ITV, Revisiones, Mantenimientos, Seguros), con preavisos automáticos y posibilidad de marcar su estado (pendiente, realizado, cancelado).
- **SMS de Avería (opcional):** Historial de avisos enviados por equipos instalados en campo, con posibilidad de filtro por cliente o proyecto. El módulo está planteado para fases posteriores del proyecto.

Diagrama Entidad-Relación



Nota sobre el diagrama entidad-relación:

Con el fin de mantener una presentación clara y comprensible, el siguiente diagrama entidad-relación (ER) muestra únicamente las entidades y sus relaciones principales dentro del sistema Innova66. Por motivos de legibilidad, no se han representado las cardinalidades ni los atributos específicos de cada entidad, centrándose únicamente en visualizar la estructura general del modelo de datos y la conexión lógica entre sus componentes. Esta decisión permite una lectura más fluida y evita la sobrecarga visual, especialmente en diagramas con alta densidad de relaciones.

Gracias a estas entidades y sus relaciones, el sistema puede gestionar de forma integral el registro de jornadas, la planificación técnica, el uso y control de materiales, la generación de albaranes listos para envío, y el seguimiento de eventos clave del negocio, todo ello con trazabilidad total y cumpliendo normativa legal.

3.2. Análisis y diseño del sistema funcional

El sistema Innova66 se divide en módulos funcionales que responden directamente a las necesidades operativas de la empresa Indasor 66 S.L. Cada módulo ha sido diseñado para digitalizar procesos previamente manuales y facilitar la gestión eficiente de los recursos técnicos, humanos y materiales. A continuación se describen los módulos implementados:

Gestión de actividades

Permite a los técnicos registrar cada intervención realizada durante su jornada laboral o en caso de urgencia fuera de jornada. Cada actividad incluye cliente, proyecto (si aplica), ubicación, fecha, hora de entrada y salida, descripción de tareas, observaciones y materiales utilizados. Estas actividades quedan registradas con trazabilidad total, permitiendo al administrador generar informes filtrados por técnico, cliente, proyecto o periodo. Además, sirven como base para la creación de albaranes.

Gestión de inventario

Este módulo centraliza el control de stock de materiales clasificados por tipo y unidad de medida. Permite registrar nuevos artículos, editar sus propiedades o darlos de baja lógica. Al registrar actividades, los materiales utilizados se descuentan automáticamente del inventario. También es posible registrar compras, que actualizan el stock y se almacenan en un historial detallado. El sistema genera alertas cuando un artículo alcanza su stock mínimo, y permite asignar niveles de prioridad a los materiales críticos.

Gestión de proyectos y calendario

Desde este módulo el administrador puede registrar y gestionar los proyectos activos, asociándolos a un cliente y añadiendo detalles como ubicación, estado y fecha de inicio. Se integra además un calendario con fechas clave vinculadas o no a proyectos, clasificadas por tipo de evento (ITV, Revisiones, Mantenimientos, Seguros). El sistema genera automáticamente alertas próximas según el tipo de evento, con preavisos específicos (30 o 60 días) y permite marcar cada evento como pendiente, realizado o cancelado.

Gestión de albaranes

Este módulo permite al administrador generar albaranes profesionales a partir de las actividades realizadas. Se agrupan por fecha, cliente y proyecto, y pueden ser sin valorar (solo materiales y horas) o valorados (con precios, descuentos y totales). El sistema genera automáticamente las líneas de materiales desde las actividades, y permite añadir manualmente una línea de mano de obra con descripción y cantidad estimada. El total se calcula automáticamente y se genera un PDF con diseño corporativo listo para impresión o envío.

Gestión de usuarios y permisos

El sistema contempla dos perfiles: técnico y administrador. Los técnicos tienen acceso únicamente a las funciones necesarias para registrar su jornada, actividades y consultar materiales o eventos. El administrador accede a todos los módulos, incluyendo inventario, proyectos, usuarios, compras, eventos, informes y albaranes. El control de permisos está garantizado tanto en backend como en frontend, cumpliendo con los principios de mínimo privilegio y seguridad de los datos.

Centralización de averías (módulo opcional)

Está prevista la futura integración con APIs externas (como Twilio) para recibir SMS generados automáticamente por equipos instalados en campo ante fallos técnicos. El sistema almacenará estos mensajes en un historial, permitirá filtrarlos por cliente o proyecto y enviará notificaciones automáticas al equipo técnico, mejorando el tiempo de respuesta ante incidencias.

Gracias a esta división modular, el sistema garantiza una gestión eficiente y escalable de los procesos clave de la empresa, manteniendo una separación clara de responsabilidades y una trazabilidad completa de la información registrada.

3.3. Análisis y diseño de la interfaz de usuario

La interfaz de usuario de Innova66 ha sido diseñada para adaptarse a los distintos perfiles de uso dentro de la empresa, diferenciando claramente entre la aplicación de escritorio (para el administrador) y la aplicación móvil (para los técnicos). El diseño se ha basado en principios de usabilidad y experiencia de usuario (UX), buscando simplicidad, claridad visual y eficiencia en entornos de trabajo técnico y operativo.

Aplicación de escritorio (rol administrador)

La versión de escritorio, orientada al administrador de la empresa, ofrece un panel principal con acceso a todos los módulos del sistema mediante una barra lateral fija. Cada módulo (inventario, clientes, proyectos, jornadas, actividades, compras, eventos, albaranes y usuarios) cuenta con vistas especializadas y formularios detallados.

Se han diseñado pantallas para registrar y editar materiales, gestionar proyectos y fechas clave del calendario, filtrar eventos o generar informes. El sistema permite generar albaranes de forma manual a partir de las actividades agrupadas, mostrando una vista previa de los materiales utilizados y permitiendo la edición de líneas con precios y descuentos.

Aplicación móvil (rol técnico)

La versión móvil está orientada a los técnicos que trabajan en campo. La interfaz está simplificada para permitir un uso rápido y directo desde dispositivos Android, incluso en situaciones con conectividad limitada.

Desde la pantalla de inicio se accede al registro de jornada diaria y al registro de actividades. Se incluyen selectores de cliente y proyecto con filtros dinámicos, selección de materiales utilizados con búsqueda por nombre o ID, y validaciones automáticas de horarios.

Además, los técnicos pueden consultar su historial de actividades, materiales disponibles y eventos próximos. La navegación se realiza mediante un drawer lateral personalizado.

El sistema también está preparado para enviar notificaciones push (mediante Firebase Cloud Messaging) en caso de eventos críticos o futuras alertas de avería.

Diseño y estilo visual

Los primeros bocetos se desarrollaron como wireframes en papel, y posteriormente se crearon mockups digitales utilizando herramientas como Figma. Se ha seguido una arquitectura visual coherente en todas las plataformas, basada en el uso de colores neutros y el color corporativo naranja (#FE9517).

La tipografía principal utilizada es Inter, seleccionada por su alta legibilidad y compatibilidad multiplataforma. Los botones, tarjetas, iconografía y componentes cumplen una guía de estilo propia para asegurar uniformidad visual en todas las pantallas y mejorar la experiencia de usuario en el uso diario.

Guía de Estilo Visual

1. Tipografía

Tipografía principal: Inter

- Moderna, legible y versátil.
- Compatible con sistemas Windows, Android y web.
- Alternativas: Roboto, SF Pro, Segoe UI.

Jerarquía:

Elemento	Estilo	Tamaño	Peso
Título de pantalla	Inter Bold	28-32 px	700
Subtítulo / secciones	Inter SemiBold	20-24 px	600

Texto de botón	Inter Medium	16-18 px	500-600
Texto principal	Inter Regular	16 px	400
Texto secundario	Inter Light	14-15 px	300-400
Etiquetas / captions	Inter Medium	12-13 px	500

2. Componentes UI

Botones

Tipo	Color de fondo	Texto	Borde	Comportamiento hover
Primario	#FE9517	Blanco	None	Sombra ligera
Secundario	Blanco	#FE9517	2px sólido #FE9517	Cambio a fondo #FFE8CC
Deshabilitado	#F9FAFB	#6B7280	None	Cursor not-allowed

Menú lateral (Sidebar)

- Fondo: #1E293B
- Íconos activos: #FFA63B
- Íconos inactivos: #6B7280
- Tipografía: Inter, 14-16 px, color blanco/naranja

Tarjetas (Cards)

- Fondo: #F9FAFB
- Borde: 1px sólido #E5E7EB
- Esquinas: redondeadas (8-12 px)

- Sombra suave (box-shadow leve)
- Texto: #111827 para títulos, #6B7280 para subtítulos

Estados / Etiquetas

Estado	Color de fondo	Texto
Aprobado	#10B981	Blanco
Alerta	#F97316	Blanco
Pendiente	#374151	Blanco
Enlace	Fondo blanco	#1D4ED8

3. Iconografía

- Usa íconos estilo Lucide, HeroIcons o Material Icons Outlined.
- Color de íconos:
 - o Activo: #FE9517
 - o Inactivo: #6B7280
- Tamaño estándar: 24 px, consistente en todo el sistema.

4. Navegación

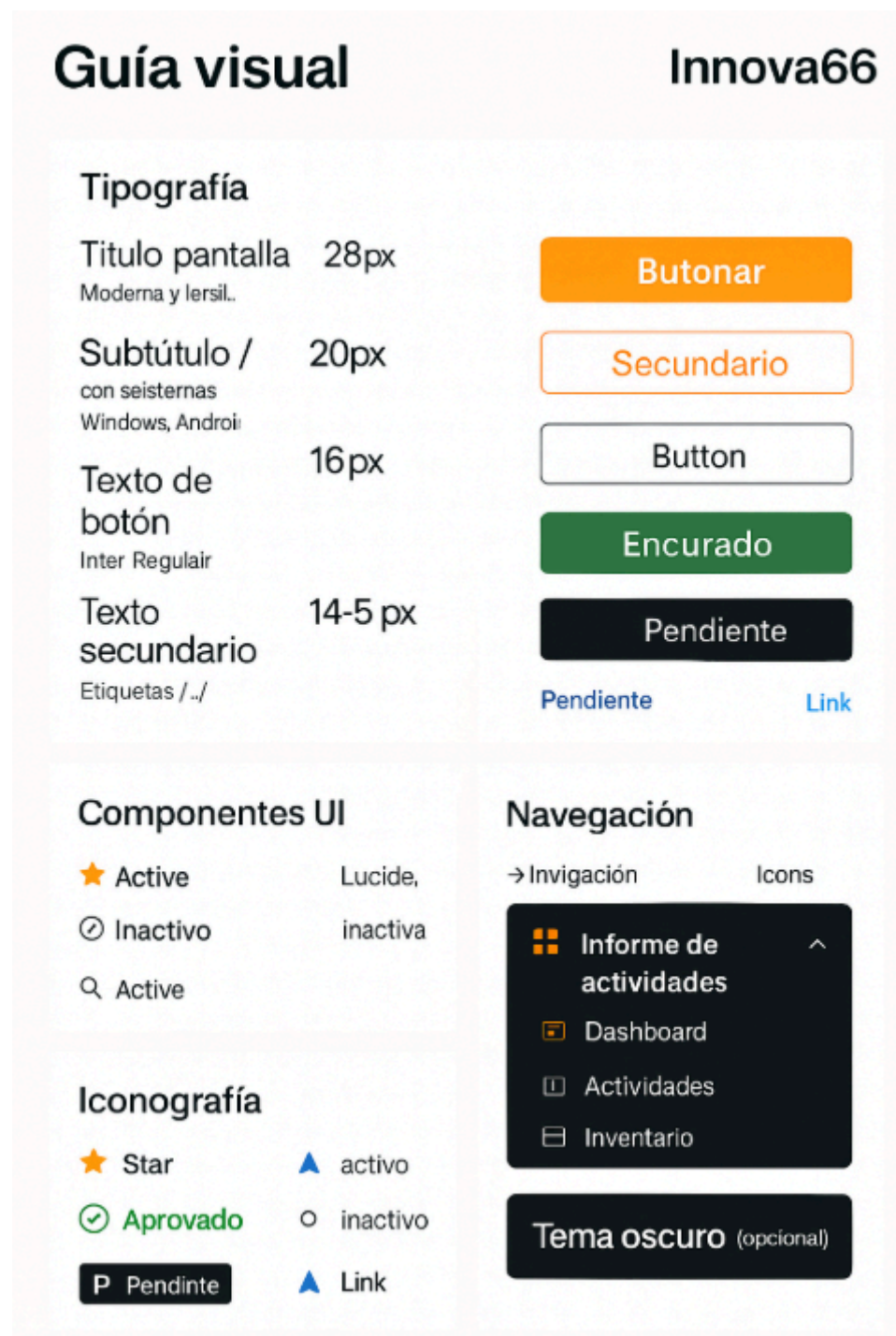
- Menú lateral permanente (modo admin en escritorio).
- Encabezado visible con título contextual y posibles acciones.
- En móvil: usar BottomNavigationBar o drawer lateral.

5. Tema oscuro (opcional futuro)

- Fondo: #0F172A

- Texto principal: #F9FAFB
- Texto secundario: #94A3B8
- Botón primario: mantener #FE9517

6. Ejemplo



Gracias a este enfoque diferenciado por perfil y entorno de uso, la aplicación Innova66 ofrece una experiencia optimizada tanto para el personal técnico como para el responsable de administración, facilitando la adopción de la herramienta y su integración en los procesos habituales de trabajo.

3.4. Diseño de la arquitectura de la aplicación

La arquitectura de la aplicación se basa en el patrón Modelo-Vista-Controlador (MVC), lo que permite separar claramente la lógica de negocio (controlador), la interfaz de usuario (vista) y el acceso a los datos (modelo). Esta separación facilita el mantenimiento, la escalabilidad y las pruebas del sistema.

3.4.1. Tecnologías/Herramientas usadas y descripción de las mismas

Tecnología/Herramienta	Descripción
Java + Spring Boot	Utilizado para el desarrollo del backend de la aplicación (API RESTful, lógica de negocio y conexión con base de datos).
Flutter	Framework multiplataforma empleado para desarrollar tanto la aplicación de escritorio (Windows) como la aplicación móvil (Android/iOS).
MySQL + MySQL Workbench	Sistema de gestión de bases de datos relacional y su herramienta gráfica para diseño y administración.
Firebase Cloud Messaging (FCM)	Servicio utilizado para enviar notificaciones push a los dispositivos móviles del equipo técnico.
Twilio (opcional)	API externa para recibir y gestionar mensajes SMS enviados desde dispositivos instalados en campo.
GitHub	Repositorio de código fuente y control de versiones del proyecto.
Postman	Herramienta para probar y documentar las APIs REST desarrolladas en el backend. Útil para validar respuestas y realizar pruebas durante el desarrollo.

3.4.2. Arquitectura de componentes de la aplicación

La aplicación se compone de los siguientes elementos clave:

- **Base de datos (MySQL)**
Aloja todas las entidades del sistema y mantiene la integridad relacional entre técnicos, actividades, materiales, proyectos y alertas.
- **Backend Java (Spring Boot)**
Desarrollado como una API RESTful, se encarga de:
 - Procesar la lógica de negocio.
 - Gestionar las solicitudes y respuestas entre frontend y base de datos.
 - Controlar permisos de usuario y validaciones.
- **Frontend (Flutter multiplataforma)**
Interfaz de usuario moderna y coherente en ambas plataformas:
 - **Escritorio (Windows):** Para el rol de administrador. Permite gestionar inventario, actividades, albaranes y calendario.
 - **Móvil (Android/iOS):** Para los técnicos. Facilita el registro de actividades, recepción de notificaciones y consulta de intervenciones.
- **Servicios externos**
 - **Twilio** (opcional) para recibir y registrar automáticamente SMS de averías en una interfaz centralizada.
- **Pruebas e integración (Postman)**
Se utilizará Postman para testear todas las rutas de la API REST, validar respuestas, simular peticiones desde los distintos roles de usuario y generar documentación técnica.

El sistema funcionará en un servidor local con Windows Server, conectado a una red local con acceso remoto controlado. Esto permitirá que los técnicos puedan acceder a la aplicación desde sus dispositivos móviles mientras están en campo, sin comprometer la seguridad ni la integridad de los datos.

4. Documento de implementación e implantación del sistema

4.1 Implementación

La implementación de la aplicación **Innova66** ha sido llevada a cabo de forma progresiva y modular, abarcando tanto el backend (desarrollado con Spring Boot) como el frontend (realizado con Flutter). A continuación se detalla el proceso seguido para implementar cada uno de los componentes del sistema, especificando las tecnologías utilizadas, la estructura del código y las decisiones técnicas adoptadas.

1. Preparación del backend (Spring Boot)

Se generó un proyecto base desde Spring Initializr, incluyendo las dependencias necesarias:

- Spring Web
- Spring Data JPA
- MySQL Driver
- Spring Security
- Validation
- Spring Boot DevTools
- Spring Boot Actuator
- Lombok

Se configuró el archivo `application.properties` con los datos de conexión a la base de datos MySQL (ubicada en servidor local Windows Server).

La estructura del backend se definió claramente en paquetes:

- `config/` (configuraciones de CORS y seguridad JWT con BCrypt)
- `models/` (entidades JPA)
- `models/request/` (DTOs de entrada)

- models/response/ (DTOs de salida)
- repository/ (interfaces JPA)
- services/ (lógica de negocio)
- api/ (controladores REST)

2. Módulo de Autenticación y Usuarios

Se comenzó creando la entidad Usuario, junto con sus relaciones (1:N con Jornada y Actividad) y los campos adicionales telefono, oficialia y rol. Se implementó el login utilizando Spring Security y BCrypt para el hash de contraseñas.

Endpoints clave:

- POST /api/usuarios/register
- POST /api/usuarios/login

El login fue ajustado para devolver un objeto Usuario completo (JSON) para facilitar la integración con Flutter.

3. Módulo de Jornadas y Actividades

Este es el núcleo operativo de los técnicos. Se implementaron las entidades Jornada, Actividad, y ActividadMaterial (relación N:M con Material).

- El técnico puede iniciar una jornada, indicando la hora de entrada, y cerrarla al finalizar.
- Se integró la posibilidad de registrar actividades sin jornada (caso de averías urgentes), haciendo el campo id_jornada nullable.
- El sistema valida que no se puedan crear jornadas duplicadas por día y técnico.

Endpoints destacados:

- POST /api/jornadas y PUT /api/jornadas/finalizar/{id}

- POST /api/actividades

4. Módulo de Materiales e Inventario

Se desarrolló un CRUD completo para materiales, incluyendo:

- Filtro por nombre o ID
- Filtro por tipo de material (dropdown)
- Alerta visual por stock bajo (comparando con stockMinimo)

Cada actividad registrada puede descontar materiales del inventario usando ActividadMaterial. El stock se ajusta automáticamente.

5. Módulo de Clientes, Contactos y Proyectos

- Los clientes pueden tener uno o varios contactos asociados.
- Los proyectos están relacionados con un cliente y pueden tener eventos.
- El sistema permite evitar proyectos duplicados por cliente mediante una query personalizada.

Entidades: Cliente, Contacto, Proyecto.

Filtros implementados:

- Por cliente en la pantalla de proyectos.
- Por estado de proyecto (pendiente, finalizado, etc.).

6. Módulo de Eventos del Calendario y Alertas

- Se implementó EventoCalendario con 4 tipos definidos: ITV, Mantenimiento, Revisiones, Seguros.

- Se creó el sistema de alertas automáticas, con entidad Alerta asociada a Material o EventoCalendario.
- Se muestran las alertas próximas en HomeAdmin agrupadas por tipo y estado.

7. Módulo de Historial de Compras

- Registro de compras por material, proveedor, fecha, precio unitario, cantidad y descuento.
- Cálculo automático de precioFinal.
- Incremento automático del stock del material.

Filtros disponibles:

- Por fecha, material y proveedor (se eliminó este último tras reunión con cliente).

8. Módulo de Albaranes

- Se redefinió la lógica: un albarán se genera manualmente por día, cliente y proyecto.
- Se generan automáticamente las líneas de materiales.
- El administrador puede añadir una línea de Mano de Obra.
- Se implementó la entidad LineaAlbaran y su historial de precios para facilitar la valoración.
- Generación de PDF con OpenPDF, incluyendo logo, cabecera, tabla de contenidos y firma del cliente.

9. Implementación del Frontend (Flutter)

Estructura de carpetas:

- data/models/ → modelos Dart sincronizados con entidades del backend.
- repositories/ → lógica de acceso a la API REST.

- providers/ → gestión de estado (usuario, jornada, materiales, etc.).
- screens/ → pantallas separadas por rol (admin, tecnico, shared).
- widgets/ → formularios reutilizables.

Características destacadas:

- Login funcional con redirección por rol.
- Registro de jornada con validaciones y aviso si hay jornada activa.
- Registro de actividades con o sin jornada.
- Gestor de materiales con filtros en tiempo real.
- CRUD de clientes, contactos, proyectos, usuarios.
- Sistema de eventos con filtros por tipo, fecha y estado.
- Pantalla de albaranes con filtros por cliente, proyecto, estado e ID.
- Generador de PDF accesible desde el detalle del albarán.

10. Control de versiones y pruebas

- Se usó GitHub para llevar el control del proyecto.
- Todo el backend fue probado con Postman.
- El frontend fue probado en emulador Android (Pixel 5) y dispositivo físico.
- Se resolvieron problemas técnicos de CORS, errores 400 por formato de fecha/hora, bucles de serialización infinita, y consistencia de datos entre entidades.

Conclusión de la implementación

La implementación de Innova66 se ha realizado siguiendo un enfoque modular, asegurando escalabilidad, trazabilidad y usabilidad. Cada módulo fue diseñado con una finalidad práctica en el día a día de Indasor 66 S.L., y la aplicación ha sido validada funcionalmente en un entorno real. El sistema está preparado para su despliegue e integración en la infraestructura de la empresa.

4.2 Pruebas

Este apartado describe las pruebas realizadas para validar el correcto funcionamiento de la aplicación Innova66 en su conjunto, tanto a nivel de backend como de frontend. Se realizaron pruebas manuales, funcionales y de integración, asegurando que cada uno de los requisitos definidos en la fase de análisis se cumple en su totalidad.

Metodología utilizada

Se adoptó un enfoque de pruebas funcionales por módulos, utilizando herramientas como Postman para validar los endpoints REST del backend y emuladores Android / Flutter para probar la experiencia de usuario desde dispositivos móviles. No se emplearon pruebas automatizadas por limitaciones de tiempo y alcance, pero se priorizó la validación exhaustiva a través de casos de uso reales aportados por la empresa Indasor 66 S.L.

Fases de prueba

1. **Pruebas unitarias manuales por módulo (backend)**
Se validaron individualmente los módulos de usuarios, jornadas, actividades, materiales, clientes, proyectos, alertas, eventos y albaranes utilizando Postman. Se verificaron todos los métodos CRUD, validaciones, relaciones entre entidades y formatos de respuesta JSON.
2. **Pruebas de integración backend + base de datos**
Cada operación fue comprobada contra MySQL Workbench para confirmar que las inserciones, actualizaciones y borrados se reflejan correctamente en la base de datos, incluyendo relaciones entre tablas y claves foráneas.
3. **Pruebas desde el frontend (Flutter)**
Se validó el flujo completo desde login hasta generación de albaranes mediante emulador Android y pruebas reales desde dispositivo físico. Se verificaron errores de red, comportamiento offline, formatos de fecha/hora, serialización JSON y navegación por roles.
4. **Pruebas de consistencia de datos**
Se analizaron posibles problemas de duplicidad, referencias nulas, ciclos infinitos de serialización y sincronización entre modelos Dart y entidades JPA. Todos ellos fueron resueltos y documentados en el diario de bitácora.
5. **Pruebas específicas de funcionalidad crítica**
 - Validación de jornada activa/no duplicada por día y técnico.
 - Registro de actividades urgentes sin jornada.
 - Generación automática del stock tras uso y reposición.
 - Alerta por stock bajo y eventos próximos.
 - Cálculo automático y edición de albaranes.
 - Generación del PDF y verificación de su contenido.

Resultado general

Todas las pruebas realizadas fueron exitosas, y se realizaron múltiples iteraciones para corregir errores detectados durante la validación. Las pruebas están documentadas en el diario de bitácora y han sido ejecutadas sobre datos reales proporcionados por Indasor.

Se incluye a continuación un resumen por módulo:

Módulo	Tipo de prueba	Herramienta	Resultado
Autenticación	Registro/Login usuario	Postman / Flutter	OK
Jornada	Inicio y cierre de jornada	Postman / Flutter	OK
Actividad	Registro con y sin jornada	Postman / Flutter	OK
ActividadMaterial	Descuento automático de stock	Postman	OK
Inventario	Búsquedas y alertas	Flutter	OK
Clientes/Contactos	CRUD completo y asociaciones	Postman / Flutter	OK
Proyectos	CRUD + validación de duplicados	Postman / Flutter	OK
Historial de compras	Cálculo y filtros avanzados	Postman / Flutter	OK
Eventos/Alertas	Generación y paneles de próximos	Postman / Flutter	OK
Albaranes	Creación, edición, PDF y valoración	Postman / Flutter	OK

5. Documento de cierre

5.1 Documento de instalación y configuración

Este apartado proporciona una guía técnica para instalar, configurar y ejecutar correctamente el sistema Innova66 tanto en su entorno de backend como de frontend. Está dirigido a desarrolladores o personal técnico que desee desplegar la aplicación en su infraestructura.

Requisitos previos

Backend (Spring Boot):

- Java JDK 17 o superior
- Maven
- MySQL Server
- MySQL Workbench
- IDE utilizado: Visual Studio Code

Frontend (Flutter):

- Flutter SDK 3.x
- Android Studio
- Emulador configurado o dispositivo físico Android

Instalación del backend

1. Clonar el repositorio:

```
git clone https://github.com/coderebootx/innova66-backend.git
```

2. Configurar la base de datos:

- Crear una base de datos en MySQL llamada innova66.
- Ajustar el archivo `src/main/resources/application.properties` con los datos de conexión:

```
spring.datasource.url=jdbc:mysql://localhost:3306/innova66
```

```
spring.datasource.username=tu_usuario
```

```
spring.datasource.password=tu_contraseña
```

3. Importar el proyecto en Visual Studio Code y compilar:

```
./mvnw clean install
```

4. Ejecutar la aplicación:

```
./mvnw spring-boot:run
```

5. Acceso al backend:

- La API REST se expone por defecto en: `http://localhost:8080`

6. Requisitos adicionales configurados durante el desarrollo:

- Librerías necesarias en `pom.xml`: Spring Web, Spring Data JPA, MySQL Driver, Spring Security, Validation, DevTools, Lombok, Spring Boot Actuator, OpenPDF (para generación de PDF).

Instalación del frontend (Flutter)

1. Clonar el repositorio:

```
git clone https://github.com/coderebootx/innova66-frontend.git
```

2. Instalar dependencias del proyecto:

```
flutter pub get
```

Estas dependencias están ya definidas en el archivo pubspec.yaml del proyecto. No se requiere instalación manual adicional tras clonar.

3. Editar archivos de configuración si es necesario:

- En lib/services/apiservice.dart, asegurar que el host apunta a 10.0.2.2 si se está usando un emulador Android, o a la IP local del backend si se accede desde un dispositivo real.

4. Ejecutar la app:

```
flutter run
```

Configuración adicional (opcional)

- **Logo de empresa:**
 - o El logo de Indasor se utiliza en la cabecera de los albaranes generados en PDF. Su ubicación debe coincidir con la ruta interna definida en el generador (se recomienda mantenerlo en el proyecto backend en carpeta /logo).
- **PDFs generados:**
 - o En la versión final de la aplicación, los albaranes en PDF se guardan localmente en el sistema de archivos del equipo, en la carpeta Documentos/albaranes/, organizada por tipo (ej: no_valorados/, valorados/).
 - o El sistema elige automáticamente esta ubicación sin intervención del usuario y no requiere configuración extra.

Con esta configuración, el sistema queda totalmente preparado para ser ejecutado en entorno local y, posteriormente, adaptado para entornos de producción si fuera necesario. Cada uno de los pasos ha sido validado durante el desarrollo y pruebas del proyecto Innova66.

5.2. Manual de usuario – Innova66

Este manual está pensado para que cualquier persona, incluso con poca experiencia en el uso de aplicaciones, pueda aprender a manejar la herramienta Innova66 paso a paso. Está dividido en dos perfiles de uso:

- Técnicos (que utilizan la app en el móvil)
- Administrador (que usa la app desde el ordenador)

A. USUARIO TÉCNICO (App móvil Android)

1. Iniciar sesión

Cuando abres la aplicación en el móvil:

- Verás una pantalla donde debes escribir tu **correo electrónico** y **contraseña**.
- Pulsa el botón **"Iniciar sesión"**.
- Si los datos son correctos, entrarás en la pantalla principal del técnico.

Si te equivocas, revisa que esté bien escrito tu correo y que no hayas dejado espacios sin querer.

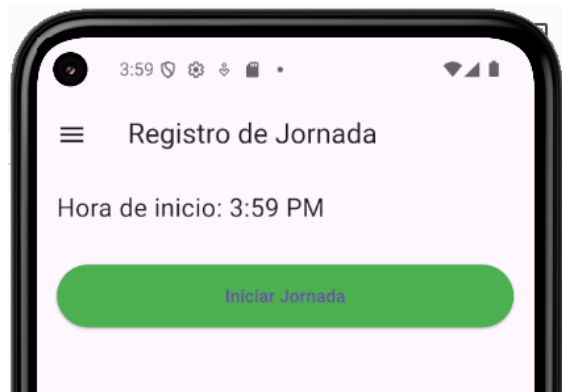


2. Registrar jornada laboral

La jornada es como "fichar" al entrar y salir del trabajo. Se hace así:

a) Iniciar jornada

- Pulsa "Registrar jornada".
- El sistema marcará automáticamente la hora de entrada.
- Puedes añadir una observación si lo deseas (por ejemplo, "empezamos a montar cuadro").
- Pulsa "Inicar Jornada".



b) Finalizar jornada

- Al acabar el día, vuelve a la pantalla de jornada.
- Pulsa "Finalizar jornada".
- Marca si has hecho pausa para comer (hay una casilla que puedes activar).
- El sistema registrará la hora de salida.

Recuerda: no puedes registrar más de una jornada por día. Si la app te lo impide, revisa si ya iniciaste una anteriormente.



3. Registrar actividad diaria

Dentro de tu jornada puedes registrar las tareas que haces. Por ejemplo, "instalación de protecciones" o "mantenimiento en nave de cliente".

a) Desde jornada activa:

- Pulsa "Nueva actividad".
- Selecciona el cliente (hay un listado, puedes escribir para buscar rápido).
- Selecciona el proyecto (opcional).
- Indica la hora de entrada y salida de esa tarea.
- Describe las tareas realizadas.
- Si tienes algo que comentar, puedes escribir en el campo observaciones.
- Pulsa "Guardar".

b) Actividad urgente (sin jornada)

Si tienes que acudir a una avería o trabajo urgente y no tienes jornada iniciada:

- Pulsa "Registrar actividad urgente".
- Aparecerá una pantalla similar, pero no necesitas jornada previa.
- Completa los mismos datos que en una actividad normal.

4. Añadir materiales usados en la actividad

Cuando estés registrando una actividad:

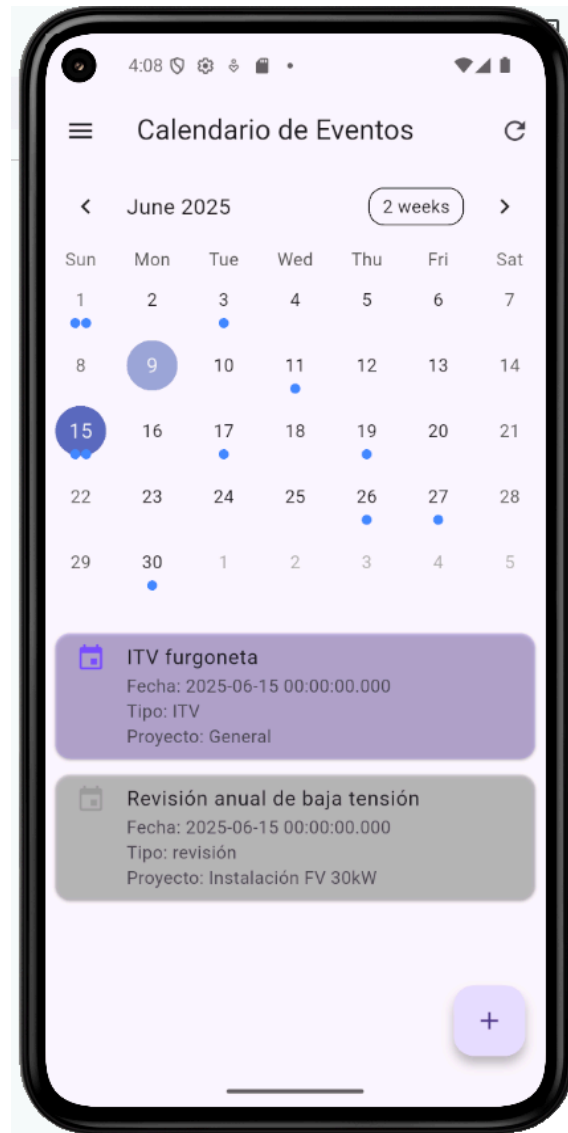
- Pulsa "Añadir material".
- Elige el material utilizado (cable, automático, contactor, etc.).
- Indica cuántas unidades usaste.
- Pulsa "Añadir material". Puedes repetir este paso para añadir más materiales.

El sistema descuenta estos materiales del inventario para llevar el control de stock.



5. Eventos del calendario

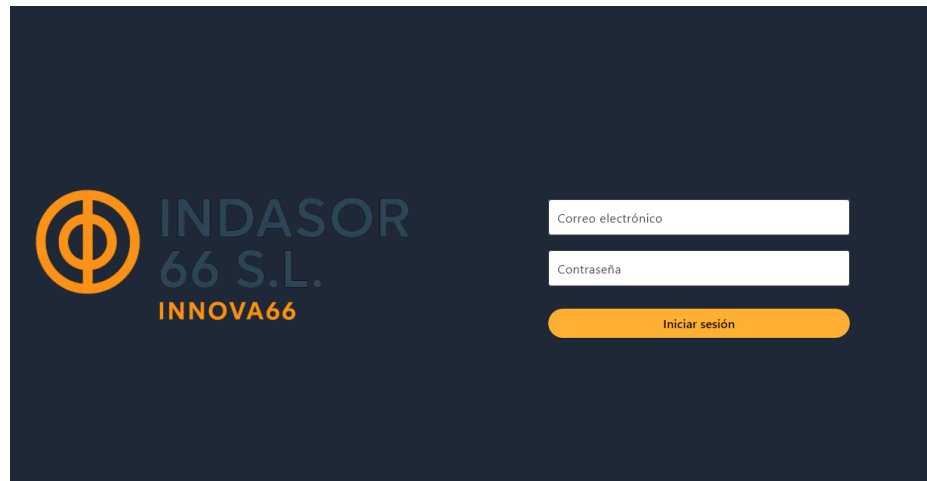
- Entra en "Eventos".
- Puedes registrar inspecciones, seguros, revisiones, etc.
- Asigna fecha, tipo (ITV, mantenimiento...) y estado (pendiente, realizado...)
- El sistema genera alertas según la fecha y tipo del evento.



B. USUARIO ADMINISTRADOR (App de escritorio Windows)

1. Iniciar sesión

- Abre la aplicación desde el ordenador.
- Introduce tu correo y contraseña.
- Pulsa "Iniciar sesión" para entrar al panel principal.



2. Panel de administración

Desde aquí puedes acceder a todos los módulos del sistema:

- Inventario
- Clientes y contactos
- Proyectos
- Generar albaranes
- Albaranes
- Eventos del calendario
- Usuarios
- Compras



3. Gestionar materiales (Inventario)

- Entra en "Inventario".
- Puedes ver la lista de materiales existentes.
- Botón "Nuevo material" para registrar uno nuevo.
- Puedes editar, eliminar o reponer cantidades.
- Los materiales tienen unidad, tipo, stock actual y mínimo.

El sistema te avisa si un material está por debajo de su stock mínimo.



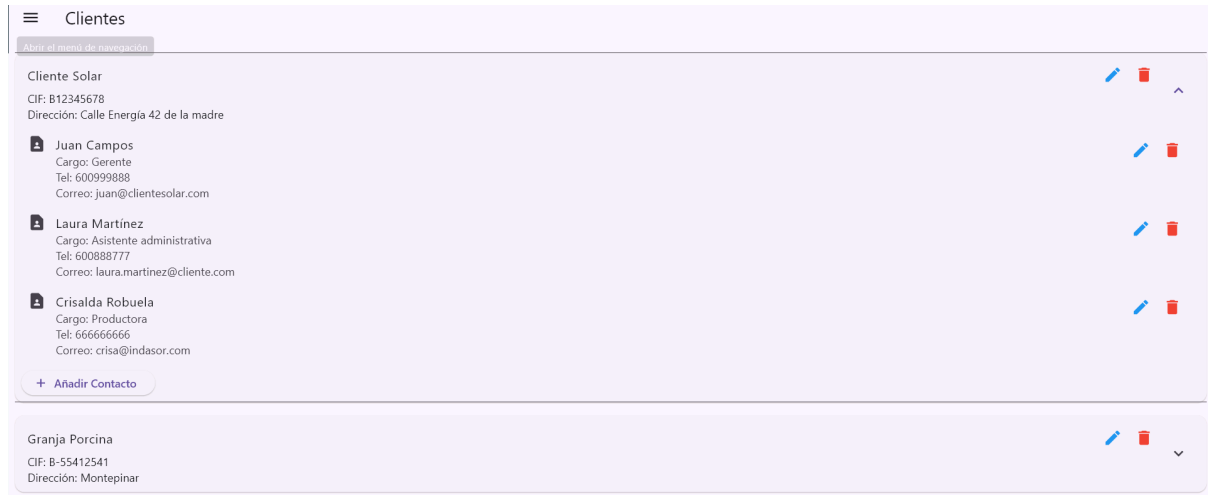
4. Registrar compras de material

- Entra en "Historial de compras".
- Pulsa "Registrar compra".
- Selecciona el material comprado.
- Introduce proveedor, fecha, cantidad, precio y descuento (si lo hay).
- Al guardar, el stock se actualiza automáticamente.



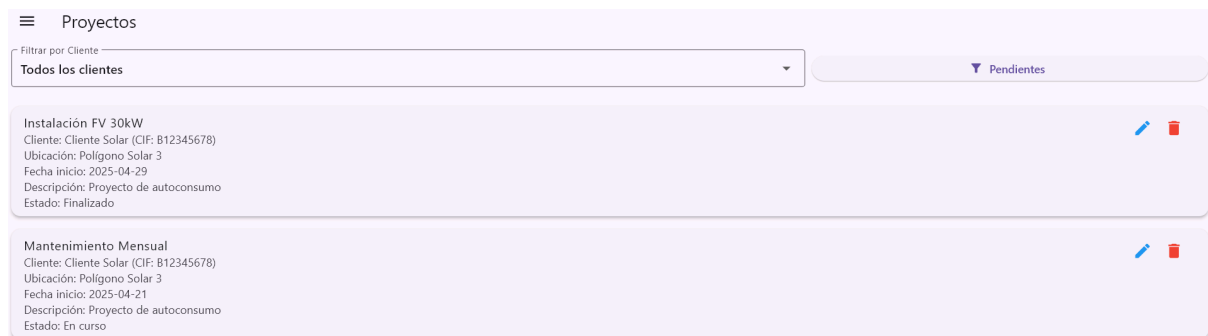
5. Gestionar clientes y contactos

- Entra en "Clientes".
- Puedes crear un nuevo cliente con nombre, CIF y dirección.
- Dentro de cada cliente puedes añadir contactos (persona, teléfono, cargo, email), pulsa sobre el cliente para desplegarlo.



6. Gestionar proyectos

- Entra en "Proyectos".
- Puedes crear uno nuevo y asociarlo a un cliente.
- Introduce nombre, descripción, ubicación, estado y fecha de inicio.



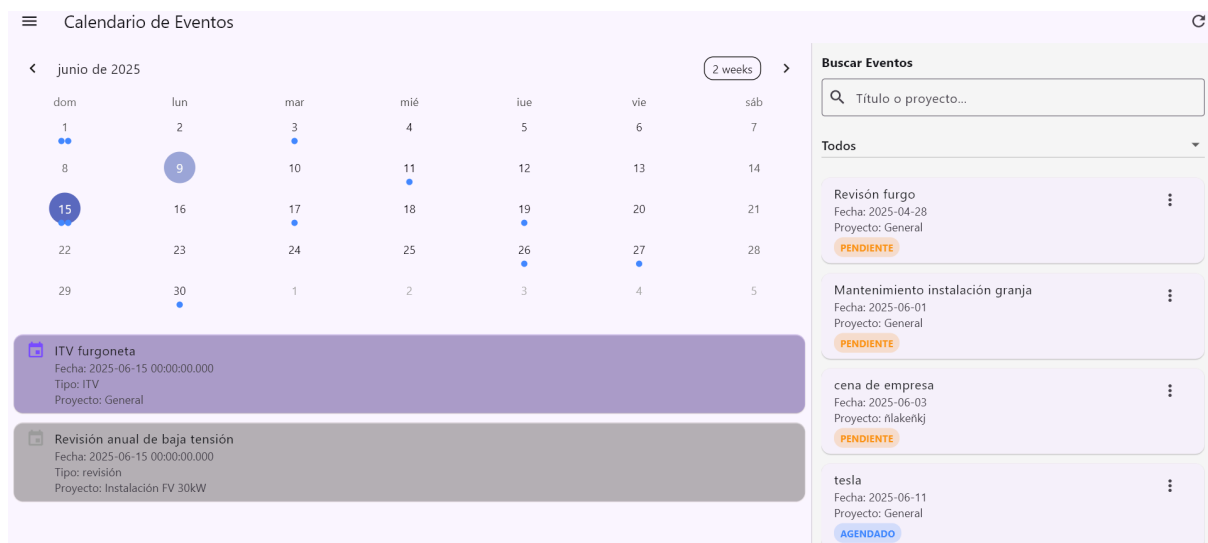
7. Eventos del calendario

En la pantalla Eventos-calendario puedes consultar todos los eventos importantes planificados:

- Se muestra un calendario mensual interactivo.

- Al seleccionar una fecha, se muestra la lista de eventos programados para ese día.
- Puedes usar el buscador para filtrar por título o proyecto.
- Hay botones para ver:
 - o Eventos próximos (por tipo: ITV, revisión, etc.)
 - o Eventos vencidos
 - o Todos los eventos
- El color del evento ayuda a identificar su tipo y estado (pendiente, realizado o cancelado).

Esta pantalla ayuda a que los técnicos no pierdan de vista las tareas legales o de mantenimiento que se aproximan.



8. Generar albaranes

- Entra en "Generar albaranes".
- Verás las actividades agrupadas por fecha, cliente y proyecto.
- Selecciona un grupo y pulsa "Generar albarán" y se desplegará edición.
- Añade una línea de mano de obra (resumen de tareas, horas trabajadas).
- Al guardar, se genera un PDF con diseño profesional sin valorar.

Generar Albarán

Granja Porcina

Proyecto: Sin proyecto

Fecha: 2025-06-02 — 1 actividades

Actividades registradas:

Juan Pérez

Tareas: Dar de comer a los cerdos

Horario: 18:17 – 20:17

Línea de Mano de Obra

Referencia (Ref.)

Descripción de tareas

Unidades (horas estimadas)

✓ Generar Albarán

Cliente Solar

Proyecto: Sin proyecto

Fecha: 2025-06-01 — 1 actividades

9. Albaranes

Entra en "Albaranes" para consultar los ya creados:

- Verás una lista de albaranes existentes, empezando por los no valorados.
- Puedes pulsar sobre cualquiera para ver sus detalles.
- En cada albarán puedes:
 - o Cambiar su estado (pendiente, aprobado, enviado) desde los tres punto.
 - o Pulsar "Editar" (símbolo del dólar) para valorar las líneas (añadir precio, descuento).
 - o Pulsar sobre "Ojo" para ver detalles y descargar el PDF de albarán ahora ya valorado.

Esta pantalla es clave para gestionar los albaranes antes de su envío o impresión final.

Listado de Albaranes	
Filtrar por estado	
<div> <div>ID: AVI25073</div> <div> <div>Cliente: Cliente Solar</div> <div>Proyecto: Instalación FV 30kW</div> <div>Fecha: 31/05/2025</div> <div>Estado: aprobado</div> </div> </div>	<div> <div></div> <div></div> <div></div> </div>
<div> <div>ID: AVI25075</div> <div> <div>Cliente: Cliente Solar</div> <div>Proyecto: Mantenimiento Mensual</div> <div>Fecha: 31/05/2025</div> <div>Estado: enviado</div> </div> </div>	<div> <div></div> <div></div> <div></div> </div>

10. Usuarios

- Entra en "Técnicos".
- Puedes crear técnicos o administradores.
- Puedes activar o desactivar usuarios (no se borran para cumplir normativa).

- Puedes filtrar para ver los desactivados desde slide esquina superior derecha.



11. Seguridad y normas

- Los técnicos no tienen acceso a funciones de administración.
- Ningún usuario puede ver datos de otros técnicos.
- La información queda guardada y no se puede modificar sin dejar rastro.

Este manual busca que cualquier usuario, aunque no tenga experiencia previa con aplicaciones, pueda usar Innova66 de forma segura, sencilla y eficaz. Cada funcionalidad ha sido pensada para ser lo más intuitiva posible, con pasos guiados y validaciones automáticas para evitar errores.

5.3 Resultados obtenidos y conclusiones

El desarrollo de la aplicación Innova66 ha permitido obtener resultados muy satisfactorios en cuanto a la digitalización de procesos internos de la empresa Indasor 66 S.L. A continuación, se detallan los logros alcanzados y las principales conclusiones extraídas tras la implementación y validación completa del sistema.

Resultados obtenidos

1. Digitalización completa del flujo de trabajo diario:

- Registro de jornadas laborales conforme a la normativa vigente.
- Registro detallado de cada intervención técnica (actividades).
- Automatización de la generación de albaranes listos para impresión o envío.

2. Control de inventario efectivo:

- Gestión centralizada de materiales.
- Descuento automático de stock según uso registrado.
- Alerta por materiales con stock bajo.

3. Gestión clara de clientes, contactos y proyectos:

- Estructura relacional clara.
- Asociación directa entre cliente, contacto, proyecto y actividad.

4. Agenda centralizada y recordatorios:

- Sistema de eventos con calendario y alertas automáticas.
- Panel de eventos próximos agrupados por tipo y estado.

5. Modularidad y trazabilidad del sistema:

- División clara por módulos: usuarios, actividades, materiales, clientes, eventos...
- Registro seguro y consultable de todas las operaciones realizadas.

6. Interfaz diferenciada según perfil:

- Aplicación adaptada a técnicos (dispositivo móvil) y administradores (escritorio).
- Flujo de uso optimizado según rol y necesidades reales.

7. Generación de documentos profesionales:

- Exportación de albaranes en formato PDF.
- Formato estructurado con logo, desglose de materiales y mano de obra.

8. Cumplimiento de objetivos y requisitos definidos:

- Todos los módulos priorizados en la fase de análisis han sido implementados y validados.

Conclusiones

- **El proyecto ha demostrado ser completamente viable** desde el punto de vista técnico, económico y funcional, adaptándose a los flujos reales de trabajo de Indasor 66 S.L.
- **El enfoque modular y progresivo** ha permitido una implementación ordenada, evitando errores de arquitectura y facilitando las pruebas.
- **La participación activa del cliente** (CEO de Indasor) ha sido clave para alinear los requisitos con la solución, validando en todo momento los avances realizados.
- **La trazabilidad y control ofrecidos por Innova66** suponen una mejora radical respecto al sistema anterior basado en papel, hojas de cálculo y procesos manuales.
- **La experiencia adquirida** durante el desarrollo ha sido fundamental para consolidar competencias técnicas (Spring Boot, Flutter, MySQL, generación de PDF, autenticación, diseño de APIs RESTful) y metodológicas (planificación, pruebas, control de versiones, documentación).

En definitiva, Innova66 cumple con todos los objetivos definidos al inicio del proyecto, y se encuentra en condiciones reales de ser desplegado y utilizado en la infraestructura de Indasor, representando una solución estable, escalable y alineada con sus necesidades operativas.

5.4 Diario de bitácora

A lo largo del desarrollo del proyecto Innova66, se ha mantenido un diario de bitácora detallado con todas las actividades realizadas, decisiones técnicas adoptadas, reuniones con el cliente, problemas encontrados y sus respectivas soluciones. Dicho documento se incluye íntegramente como anexo al final de esta memoria.

A continuación, se presenta un resumen ejecutivo estructurado por fases, destacando los hitos más importantes y su evolución:

Fase inicial (abril 2025)

- Definición de requisitos con el CEO de Indasor.

- Creación de la base del backend con Spring Boot (configuración de seguridad, JWT, JPA, MySQL).
- Implementación inicial del módulo de usuarios con login funcional.
- Diseño de la arquitectura de carpetas para backend y frontend.

Fase de desarrollo principal (abril - mayo 2025)

- Implementación de los módulos clave: jornadas, actividades, materiales, clientes, contactos y proyectos.
- Desarrollo de la lógica de stock con descuentos automáticos y alertas.
- Registro de jornadas conforme a normativa española.
- Integración completa con Flutter, login desde app móvil y navegación según rol.
- Implementación del calendario de eventos y alertas asociadas.
- Creación del historial de compras y filtros avanzados.
- Validación de funcionalidades con Postman y pruebas desde emulador Android.

Fase de cierre funcional (finales de mayo 2025)

- Reestructuración total del sistema de generación de albaranes.
 - Generación manual basada en agrupación por día, cliente y proyecto.
 - Línea de mano de obra editable por el administrador.
 - Generación de PDF profesional con logo y estructura clara.
- Mejora UX/UI en pantalla de técnicos y validaciones lógicas.
- Integración de filtros dinámicos y vista colapsada de históricos en frontend.

Fase final y revisión (junio 2025)

- Verificación completa del flujo desde login hasta generación de documentos.
- Pruebas funcionales completas en entorno real.
- Redacción de manuales, documentación técnica y anexo del diario completo.

- Revisión integral del funcionamiento de cada módulo y comprobación exhaustiva de todas las pantallas de la aplicación, asegurando coherencia visual, operativa y funcional en todos los flujos del sistema.

Este seguimiento detallado ha permitido garantizar la coherencia técnica del proyecto, identificar mejoras continuas durante su evolución y documentar el proceso completo de forma transparente y verificable. La bitácora ha sido una herramienta clave de planificación y control durante todas las fases del proyecto Innova66.

6. Bibliografía

A continuación, se presenta una recopilación de fuentes de información utilizadas como apoyo durante el desarrollo del proyecto Innova66. Estas referencias han sido fundamentales tanto para la toma de decisiones técnicas como para la implementación de funcionalidades específicas y buenas prácticas de programación.

Documentación oficial y frameworks:

- **Spring Boot Documentation.** (<https://docs.spring.io/spring-boot/docs/current/reference/html/>)
 - Guía principal para la configuración del backend, creación de APIs RESTful, seguridad con Spring Security, y uso de JPA con MySQL.
- **Flutter Documentation.** (<https://docs.flutter.dev>)
 - Fuente oficial para el desarrollo de la interfaz de usuario, navegación, conexión con backend y gestión de estado con Provider.
- **Dart Language Tour.** (<https://dart.dev/guides/language/language-tour>)
 - Para comprender la sintaxis y características principales del lenguaje Dart, utilizado en el frontend.
- **MySQL 8.0 Reference Manual.** (<https://dev.mysql.com/doc/refman/8.0/en/>)
 - Para la definición de la base de datos, consultas SQL, configuración de claves primarias y foráneas.

Tutoriales y comunidades de desarrolladores:

- **Baeldung.** (<https://www.baeldung.com/>)
 - Recurso valioso para ejemplos concretos de Spring Boot, validaciones, configuración de seguridad, y manejo de errores.
- **Stack Overflow.** (<https://stackoverflow.com>)
 - Utilizado como fuente de consulta para resolver errores puntuales, ejemplos de código y recomendaciones de buenas prácticas.
- **Medium – Flutter Community.** (<https://medium.com/flutter-community>)
 - Artículos y guías paso a paso sobre diseño de interfaces, integración con APIs REST y uso de widgets personalizados.

Generación de documentos y PDFs:

- **OpenPDF GitHub Repository.** (<https://github.com/LibrePDF/OpenPDF>)
 - Proyecto utilizado para la generación de albaranes en formato PDF dentro del backend.
- **iText (archivado).** (<https://itextpdf.com/en/resources/documentation>)
 - Aunque no se utilizó directamente, su documentación fue útil para comprender conceptos básicos de composición de PDFs.

Diseño y UX/UI:

- **Figma.** (<https://www.figma.com>)
 - Utilizado para la creación de prototipos iniciales de las pantallas móviles y visualización del flujo de navegación.
- **Material Design Guidelines.** (<https://m3.material.io/>)
 - Referencia para la implementación de componentes visuales y experiencia de usuario coherente en la app Flutter.

Normativa legal y empresarial:

- **Real Decreto-ley 8/2019, de medidas urgentes de protección social.**
(<https://www.boe.es/eli/es/rdl/2019/03/08/8/con>)
 - Norma legal utilizada como base para la implementación del control de jornada laboral.
- **LOPDGDD - Ley Orgánica 3/2018.**
(<https://www.boe.es/buscar/act.php?id=BOE-A-2018-16673>)
 - Para cumplir con la normativa de protección de datos en la gestión de usuarios y registro de información sensible.

Estas fuentes han sido seleccionadas por su fiabilidad, calidad técnica y relevancia directa con los componentes del proyecto. Su correcta interpretación y aplicación han permitido garantizar la calidad del software y la adecuación funcional del sistema a las necesidades reales de la empresa Indasor 66 S.L.

7. Anexos

A continuación, se detalla el contenido que se incorpora como anexos a esta memoria. Los anexos proporcionan una visión complementaria y detallada del desarrollo de la aplicación Innova66, incluyendo documentación técnica, evidencias gráficas y soporte legal o funcional.

Anexo I – Modelo entidad-relación (MER)

- Esquema completo de la base de datos en MySQL.
- Muestra entidades, atributos, relaciones y claves primarias/foráneas.

Anexo II – Endpoints de la API REST

- Tabla de rutas organizadas por módulo: verbo HTTP, descripción, parámetros, respuesta esperada.
- Útil para desarrolladores futuros o tareas de mantenimiento.

Anexo III – Licencia de uso

- Documento que define los derechos de uso del software desarrollado.
- Incluye permisos de uso, modificación, distribución y atribución.

Anexo IV – Diario de bitácora completo

- Registro cronológico de todas las actividades realizadas durante el proyecto.

Incluye avances semanales, problemas detectados, soluciones aplicadas y validaciones.

La inclusión de estos anexos garantiza la trazabilidad del trabajo realizado, ofreciendo soporte documental que complementa la memoria técnica y respalda la calidad y completitud del proyecto **Innova66**.

Anexo I – Modelo Entidad-Relación (MER)

Este anexo describe el diseño lógico de la base de datos del sistema Innova66, incluyendo todas las entidades implementadas, sus atributos principales, relaciones entre tablas y claves primarias/foráneas, basadas directamente en las clases Java utilizadas en el backend desarrollado con Spring Boot y JPA.

1. Entidades y atributos principales**Usuario**

- **id_usuario** (PK)
- nombre
- email
- teléfono
- oficialia
- rol (técnico / administrador)
- password_hash
- activo (boolean)

Jornada

- **id_jornada** (PK)
- id_usuario (FK → Usuario)
- fecha
- hora_inicio
- hora_fin
- comio (boolean)
- observaciones_generales
- jornada_cerrada (boolean)

Actividad

- **id_actividad** (PK)
- id_jornada (FK → Jornada)
- id_usuario (FK → Usuario)
- id_cliente (FK → Cliente)
- id_proyecto (FK → Proyecto) (*opcional*)
- id_albaran (FK → Albaran) (*opcional*)
- fecha
- hora_entrada
- hora_salida
- descripcion_tareas
- observaciones

ActividadMaterial

- **id_actividad_material** (PK)

- id_actividad (FK → Actividad)
- id_material (FK → Material)
- cantidad_utilizada

Material

- **id_material** (PK, tipo String)
- nombre
- descripcion
- unidad_medida
- tipo_material
- cantidad_stock
- stock_minimo

HistorialCompra

- **id_compra** (PK)
- id_material (FK → Material)
- proveedor
- fecha
- precio_unitario
- descuento (*nullable*)
- precio_final
- cantidad
- descripcion

Cliente

- **id_cliente** (PK)

- nombre
- CIF
- direccion

Contacto

- **id_contacto** (PK)
- id_cliente (FK → Cliente)
- nombre_contacto
- cargo
- teléfono
- correo

Proyecto

- **id_proyecto** (PK)
- id_cliente (FK → Cliente)
- nombre
- ubicación
- descripción
- fecha_inicio
- estado

EventoCalendario

- **id_evento** (PK)
- id_proyecto (FK → Proyecto) (*opcional*)
- titulo_evento
- descripcion

- fecha_evento
- tipo_evento
- estado

Alerta

- **id_alerta** (PK)
- tipo_alerta
- descripcion
- fecha
- estado
- id_material (FK → Material)
- id_evento (FK → EventoCalendario)

Albaran

- **id_albaran** (PK, tipo String manual)
- fecha_generacion
- estado
- ruta_pdf
- total_calculado
- tipo_albaran
- valorado (boolean)

LineaAlbaran

- **id** (PK)
- id_albaran (FK → Albaran)
- ref

- descripcion
- unidad
- cantidad
- precio_unitario
- descuento (*nullable*)
- total_calculado

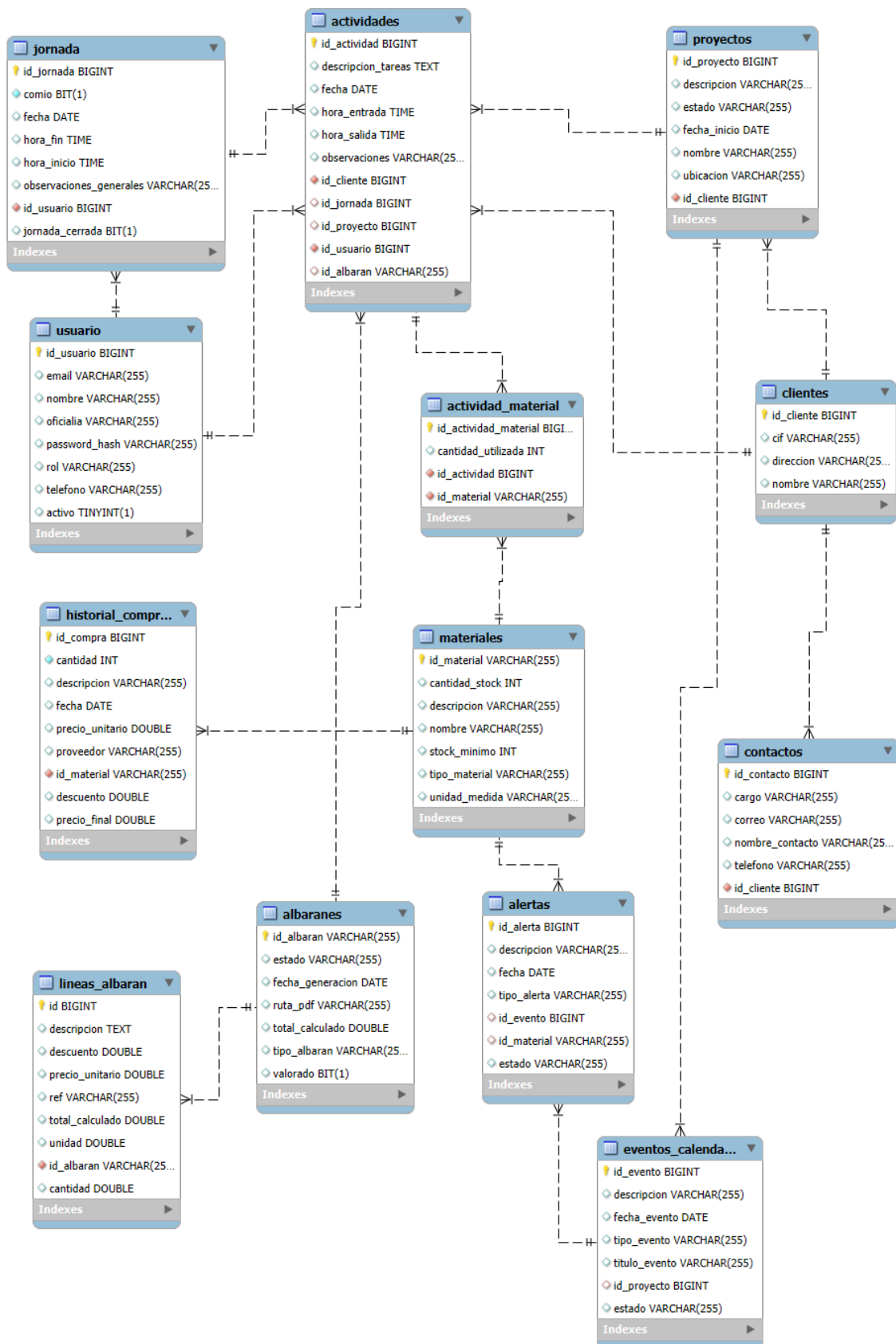
2. Relaciones clave

- **Usuario – Jornada: 1:N**
- **Jornada – Actividad: 1:N**
- **Usuario – Actividad: 1:N**
- **Cliente – Proyecto / Actividad / Contacto: 1:N**
- **Proyecto – Actividad / Evento: 1:N**
- **Actividad – ActividadMaterial – Material: N:M** (resuelta con entidad intermedia)
- **Material – HistorialCompra / Alerta: 1:N y 1:1**
- **Actividad – Albaran: N:1**
- **Albaran – LineaAlbaran: 1:N**

3. Observaciones del diseño

- Todas las claves primarias han sido definidas con identificadores únicos autogenerados, salvo en los casos especiales de Material y Albaran, donde se usan IDs personalizados.
- La base de datos está normalizada en 3FN.
- El modelo ha sido validado con MySQL Workbench y adaptado para evitar errores de serialización JSON en las relaciones bidireccionales.

Acompañando a este documento se incluye el diagrama MER completo exportado desde MySQL Workbench, en formato imagen y PDF.



Anexo II – Endpoints de la API REST

Este anexo recopila los principales endpoints disponibles en la API REST de Innova66, organizados por módulo. Se especifican las rutas, el tipo de operación (verbo HTTP), su propósito funcional y los parámetros requeridos. Esta documentación es especialmente útil para desarrolladores que requieran mantener o ampliar el sistema.

Módulo: Actividades

- POST /api/actividades → Crear nueva actividad
- GET /api/actividades/usuario/{id} → Obtener actividades por usuario
- GET /api/actividades/jornada/{id} → Obtener actividades por jornada
- GET /api/actividades/usuario/{id}/fecha/{fecha} → Por usuario y fecha
- GET /api/actividades/cliente/{id} → Por cliente
- GET /api/actividades/proyecto/{id} → Por proyecto
- GET /api/actividades/fecha/{fecha} → Por fecha (admin)
- GET /api/actividades/agrupaciones → Agrupar por fecha, cliente y proyecto
- GET /api/actividades/para-albaran → Para generación de albaranes

Módulo: ActividadMaterial

- GET /api/actividad-material/actividad/{idActividad} → Ver materiales usados en actividad
- GET /api/actividad-material/material/{idMaterial} → Ver actividades que usan un material

Módulo: Albaranes

- POST /api/albaranes/generar-desde-actividades → Crear albarán desde actividades
- GET /api/albaranes/{id} → Buscar albarán por ID
- DELETE /api/albaranes/{id} → Eliminar albarán

- PUT /api/albaranes/{id}/estado → Actualizar estado
- GET /api/albaranes/{id}/pdf → Descargar PDF

Módulo: Alertas

- GET /api/alertas/tipo/{tipo} → Por tipo de alerta
- GET /api/alertas/material/{idMaterial} → Por material
- GET /api/alertas/evento/{idEvento} → Por evento
- GET /api/alertas/fecha → Por fecha
- GET /api/alertas/proximas → Alertas futuras
- GET /api/alertas/vencidas → Alertas vencidas
- PUT /api/alertas/{id}/resolver → Marcar como resuelta

Módulo: Clientes

- GET /api/clientes/{id} → Buscar cliente por ID
- PUT /api/clientes/{id} → Editar cliente
- DELETE /api/clientes/{id} → Eliminar cliente
- GET /api/clientes/existe/nombre → Verificar duplicado por nombre
- GET /api/clientes/existe/cif → Verificar duplicado por CIF

Módulo: Contactos

- PUT /api/contactos/{id} → Actualizar contacto
- DELETE /api/contactos/{id} → Eliminar contacto
- GET /api/contactos/{id} → Obtener contacto por ID
- GET /api/contactos/cliente/{idCliente} → Todos los contactos de un cliente

Módulo: Eventos

- PUT /api/eventos/{id}/estado → Modificar estado del evento
- GET /api/eventos/tipo/{tipo} → Buscar por tipo
- GET /api/eventos/proyecto/{idProyecto} → Buscar por proyecto
- GET /api/eventos/fecha → Buscar por fecha exacta
- GET /api/eventos/entre-fechas → Buscar entre fechas
- GET /api/eventos/proximos → Eventos futuros
- GET /api/eventos/proximos/pendientes → Pendientes próximos
- GET /api/eventos/vencidos → Vencidos
- GET /api/eventos/vencidos/pendientes → Vencidos pendientes
- GET /api/eventos/entre-fechas/pendientes → Pendientes entre fechas

Módulo: Compras

- GET /api/compras/material/{idMaterial} → Compras por material
- GET /api/compras/material/{idMaterial}/ordenadas → Compras por material ordenadas
- GET /api/compras/fecha → Compras entre fechas
- GET /api/compras/proveedor → Buscar por proveedor
- GET /api/compras/proveedor-fechas → Por proveedor + fechas

Módulo: Jornadas

- PUT /api/jornadas/finalizar/{id} → Finalizar jornada
- GET /api/jornadas/usuario/{id} → Jornadas de un usuario
- GET /api/jornadas/usuario/{id}/fecha/{fecha} → Verificar duplicado
- GET /api/jornadas/fecha/{fecha} → Jornadas por fecha
- GET /api/jornadas/usuario/{id}/activa → Jornada activa actual

Módulo: Líneas de albarán

- POST /api/lineas-albaran/{idAlbaran} → Registrar múltiples líneas
- GET /api/lineas-albaran/{idAlbaran} → Ver líneas por albarán
- PUT /api/lineas-albaran/{id} → Modificar línea
- DELETE /api/lineas-albaran/{id} → Eliminar línea

Módulo: Materiales

- PUT /api/materiales/{id} → Editar material
- GET /api/materiales/{id} → Obtener por ID
- DELETE /api/materiales/{id} → Eliminar
- GET /api/materiales/buscar/nombre → Buscar por nombre
- GET /api/materiales/buscar/id → Buscar por ID (prefijo)
- GET /api/materiales/stock/bajo → Ver materiales con stock bajo
- GET /api/materiales/buscar/tipo → Buscar por tipo

Módulo: Proyectos

- PUT /api/proyectos/{id} → Editar proyecto
- DELETE /api/proyectos/{id} → Eliminar
- GET /api/proyectos/{id} → Obtener proyecto
- GET /api/proyectos/cliente/{idCliente} → Proyectos por cliente
- GET /api/proyectos/buscar → Buscar por nombre
- PUT /api/proyectos/{id}/estado → Actualizar estado
- GET /api/proyectos/estado/{estado} → Filtrar por estado
- GET /api/proyectos/existe → Verificar duplicado por nombre y cliente

Módulo: Usuarios

- POST /api/usuarios/registro → Registrar nuevo usuario
- POST /api/usuarios/login → Login con email y contraseña
- GET /api/usuarios/{id_usuario} → Obtener usuario
- PUT /api/usuarios/{id_usuario} → Actualizar usuario
- DELETE /api/usuarios/{id_usuario} → Eliminar usuario
- PUT /api/usuarios/{id_usuario}/estado → Cambiar estado (activo/inactivo)

Cada uno de estos endpoints ha sido probado y validado durante el desarrollo del sistema **Innova66**, cumpliendo con la estructura RESTful estándar para facilitar su uso e integración con el frontend Flutter u otras aplicaciones futuras.

Anexo III – Licencia de uso

Este documento define los derechos de uso del software desarrollado, los límites de su explotación, y las condiciones bajo las cuales puede compartirse o modificarse. A continuación se detalla cada uno de los aspectos clave para su comprensión y cumplimiento.

1. Nombre del proyecto

Innova66 – Sistema interno de gestión técnica y administrativa para la empresa Indasor 66 S.L. desarrollado como Trabajo Fin de Grado. Representa una solución personalizada para el entorno laboral y organizativo de la empresa.

2. Titularidad

La propiedad intelectual del software y todo su código fuente pertenece exclusivamente al desarrollador del proyecto, quien lo ha creado en el marco de su formación académica. El cliente (Indasor 66 S.L.) tiene derecho al uso de la aplicación compilada, pero no adquiere derechos sobre el código fuente salvo pacto expreso por escrito. Esta condición protege la autoría del proyecto y evita modificaciones no autorizadas.

3. Acceso al código fuente

Este software **no es de código abierto**. No se facilita el acceso al código fuente ni se permite su inspección técnica. El cliente recibe únicamente:

- La versión funcional compilada para su entorno de uso.
- Documentación técnica y manuales necesarios para su instalación y utilización.

Cualquier solicitud de acceso, revisión o auditoría del código deberá gestionarse directamente con el desarrollador.

4. Permisos de uso

Se concede al cliente el derecho a:

- Instalar la aplicación en ordenadores y dispositivos móviles utilizados por la empresa.
- Utilizarla de forma ilimitada en el tiempo, dentro del entorno interno de Indasor 66 S.L.

No se permite:

- Compartir la aplicación con terceros ajenos a la empresa.
- Venderla o sublicenciarla.
- Usarla en empresas externas o fuera del ámbito de aplicación previsto.

5. Permisos de modificación

No se permite realizar cambios, ingeniería inversa, ni descompilar la aplicación entregada. Esta restricción garantiza la estabilidad del sistema y evita vulnerabilidades. Cualquier mejora, adaptación o ampliación deberá solicitarse directamente al desarrollador.

La arquitectura del sistema ha sido diseñada para ser ampliable en futuras versiones, siempre que estas sean ejecutadas por el autor o bajo su supervisión.

6. Uso comercial

El sistema puede utilizarse de manera **comercial únicamente dentro de Indasor 66 S.L.** Esto significa que puede emplearse para mejorar la eficiencia operativa de la empresa, generar documentos oficiales (albaranes, informes, etc.), y formar parte de sus procesos internos. Sin embargo, **no puede ser revendido, distribuido o ofrecido como producto comercial a terceros.**

7. Atribución

En caso de que la aplicación se presente en foros, ferias, publicaciones, o en el marco de colaboraciones institucionales, se deberá mencionar claramente el nombre del desarrollador como autor del sistema.

Ejemplo de atribución: “Aplicación Innova66 desarrollada por [Nombre del alumno] como parte de su Trabajo Fin de Grado (DAM).”

8. Consideraciones adicionales

- El desarrollador se reserva el derecho a conservar una copia del sistema con fines de portafolio profesional, lo que puede incluir demostraciones no funcionales en entrevistas o presentaciones académicas.
- Esta licencia podrá modificarse en el futuro mediante acuerdo firmado entre ambas partes si se acuerdan nuevas condiciones de explotación, mantenimiento o cesión.

9. Futuras ampliaciones del sistema

El sistema Innova66 ha sido diseñado con una arquitectura modular y escalable, lo que permite su evolución a medio y largo plazo en función de las necesidades de Indasor 66 S.L. A continuación se detallan algunas posibles ampliaciones ya previstas o propuestas:

- **Integración de alertas SMS:** Incorporación de una API de terceros (como Twilio) que permita recibir notificaciones automáticas por parte de dispositivos instalados en campo, con registro de mensajes, filtrado y notificaciones al personal técnico.
- **Gestión documental:** Módulo para adjuntar archivos a proyectos o actividades, como planos eléctricos, certificados, fichas técnicas, etc.
- **Control de vehículos:** Registro y mantenimiento de la flota de vehículos de la empresa, incluyendo ITV, seguros, revisiones y kilometraje.
- **Panel de informes personalizados:** Visualización gráfica de datos (por técnico, cliente, proyecto) mediante dashboards e informes exportables (PDF, Excel).

- **Gestión de tareas internas:** Un calendario o bandeja de entrada donde el administrador pueda asignar tareas específicas a los técnicos y hacer seguimiento de su cumplimiento.
- **Control de acceso físico y fichaje geolocalizado:** Uso de GPS o lectores NFC para verificar ubicación y horario real del técnico al iniciar jornada o actividad.
- **Historial de mantenimiento de instalaciones:** Registro cronológico de todas las intervenciones sobre una misma instalación (vinculadas al cliente o proyecto).

Estas ampliaciones se implementarán en nuevas versiones del sistema conforme se evalúen las prioridades internas, el presupuesto disponible y el tiempo de desarrollo requerido. El sistema actual ha sido diseñado para que estas funciones se integren sin afectar la estructura ya implementada.

Anexo IV – Diario de bitácora

20-21/04/2025

Descargar desde Spring Initializr un proyecto de Maven capaz de conectar a nuestra base de datos MySQL, exponiendo una API RESTful y siguiendo un patrón MVC, añadiendo para ello dependencias esenciales como:

- Spring Web → para crear controladores REST.
- Spring Data JPA → para trabajar con la base de datos MySQL mediante entidades y repositorios.
- MySQL Driver → el conector con la base de datos.
- Spring Boot DevTools → reinicio automático en desarrollo.
- Spring Security → añadir roles y login.
- Validation → útil para validar datos en DTOs.
- Spring Boot Actuator → para monitorizar el backend.
- Lombok → para reducir el código en las entidades y controladores.

Configuración de application.properties para la conexión de nuestra app con MySQL.

Creación de estructura para trabajar el backend: api, config, models, request, response, repository y services.

Con la estructura definida pasamos a implementar lo siguiente:

Módulo Usuarios y Autenticación:

- Crear entidad Usuario y su tabla.
- Crear DTOs para registro y login.
- Implementar autenticación básica, con texto plano.
- Probar la creación de usuarios y login desde Postman.

Realizamos la entidad Usuario como modelo para el resto de las entidades, para ello creamos la clases:

- models/Usuario.java → Usuario y sus propiedades, con nombre de la tabla a crear, nombre de las columnas, clave primaria y relaciones con otras entidades.
- repository/UsuarioRepository.java → Interfaz para realizar búsquedas, en este caso findByEmail.
- service/UsuarioService.java → Nos permite registrar un nuevo usuario, realizar búsquedas por id o email y la autenticación.
- models/request/UsuarioRegisterRequest.java → DTO para el registro
- models/request/UsuarioLoginRequest.java → DTO para el login
- api/UsuarioController.java → Punto de entrada a la API REST, con los endpoints necesarios, en este caso, registro y login.

Comprobación mediante postman, el correcto registro y posterior login, funcionando correctamente a dichas peticiones.

Implementación de BCryptPasswordEncoder para evitar guardar contraseñas en texto plano, aplicando un hash a la contraseña antes de guardarla.

Para ello creamos SecurityConfig.java, creando un bean para inyectar en nuestros servicios.

Problema: al importar librería de BCryptPasswordEncoder nos da fallo

Solución : añadir al archivo pom.xml dicha dependencia que no se descargo desde Initilizr.

Problema: error 401 Unauthorized desde postman

Solución: Al activar Spring Security bloquea acceso a los endpoints, por lo tanto, hay que editar SecurityConfig.java para permitir acceso a los endpoints necesarios en función de las entidades que tengamos.

Módulo Actividades + Jornadas

- Es el núcleo del día a día de los técnicos.
- Desde aquí se genera luego el albarán.
- Necesidad de registrar fechas, tareas, entrada/salida, observaciones, y materiales usados.

Entidades:

- Jornada (una por técnico y día)
- Actividad (varias por jornada)
- Actividad_Material (relación N:M con Material)

Funcionalidades:

- Crear jornada al empezar el día (técnico)
- Añadir actividades con:
 - Proyecto, cliente, hora entrada/salida, descripción, observaciones
 - Materiales utilizados
- Guardar y consultar por técnico, fecha, cliente, etc.

JornadaRepository:

- findByUsuarioAndFecha(...) → Para saber si ya tiene jornada creada ese día
- findByFecha(...) → Por si el admin quiere ver todo lo que se hizo en un día
- findByUsuario(...) → Para listar jornadas de un técnico

JornadaService:

- Crear una nueva jornada.
- Consultar jornadas por usuario.
- Consultar jornadas por fecha.
- Verificar si ya existe una jornada para un usuario y fecha determinada (evitar duplicados).

JornadaRequest:

El DTO debe incluir:

- idUsuario (quién está creando la jornada)
- fecha (día de la jornada)
- horaInicio, horaFin (horario trabajado)
- comio (booleano)
- observacionesGenerales (texto opcional)

JornadaController:

- POST /api/jornadas → Crear nueva jornada
- GET /api/jornadas/usuario/{id} → Listar jornadas por técnico
- GET /api/jornadas/usuario/{id}/fecha/{fecha} → Verificar si ya existe jornada ese día
- GET /api/jornadas/fecha/{fecha} → Ver todas las jornadas de un día

Actividad.java:

Relacionada con:

- Jornada (muchas actividades por jornada)
- Usuario (quién la realiza)
- Cliente
- Proyecto
- Material (a través de Actividad_Material)

ActividadRepository:

Consultar actividades por:

- Técnico
- Jornada
- Cliente
- Proyecto
- Fecha

ActividadService:

Registrar una nueva actividad

Obtener actividades por:

- Técnico (Usuario)
- Jornada (Jornada)
- Cliente
- Proyecto
- Fecha

ActividadRequest:

El DTO debe contener:

- idJornada
- idUsuario
- idCliente
- idProyecto
- fecha
- horaEntrada
- horaSalida
- descripcionTareas
- observaciones

ActividadController:

- POST /api/actividades → Registrar una actividad
- GET /api/actividades/usuario/{id} → Ver actividades de un técnico
- GET /api/actividades/jornada/{id} → Ver actividades de una jornada
- GET /api/actividades/cliente/{id}
- GET /api/actividades/proyecto/{id}
- GET /api/actividades/fecha/{fecha} (por si se quiere un resumen del día completo)

Como empezamos a tener dependencias, no podemos comprobar el correcto funcionamiento hasta no tener implementadas las entidades que intervienen.

ClienteRepository:

- clienteRepository.save(cliente) → guardar o actualizar
- clienteRepository.findById(id) → buscar por ID
- clienteRepository.findAll() → obtener todos los clientes
- clienteRepository.deleteById(id) → eliminar cliente
- clienteRepository.existsByCif(...) → verificar duplicados

ClienteService:

- Registrar nuevo cliente
- Buscar cliente por ID
- Listar todos los clientes
- Verificar existencia por nombre o CIF (opcional)

ProyectoRepository:

- findAll() → obtener todos los proyectos
- findById(id) → buscar uno concreto
- findByCliente(...) → todos los proyectos de un cliente específico
- findByNombreContainingIgnoreCase(...) → buscar por texto parcial en el nombre

ProyectoService:

- Registrar un nuevo proyecto
- Buscar por ID
- Listar todos
- Buscar por cliente
- Buscar por nombre parcial

ContactoRepository:

- save(contacto) → guardar o actualizar un contacto
- findById(id) → buscar un contacto por su ID
- findAll() → listar todos los contactos
- findByCliente(cliente) → obtener todos los contactos de un cliente

ContactoService:

- Registrar un nuevo contacto
- Buscar contacto por ID
- Obtener todos los contactos de un cliente
- Listar todos los contactos

Con lo que tenemos implementado podemos probar con postman:

- Clientes → Crear y listar
- Contactos → Crear y consultar por cliente
- Proyectos → Crear y consultar
- Jornadas → Crear y consultar
- Actividades → Crear y consultar si todo lo demás existe

Por ello debemos seguir dicho orden para la prueba ya que es una base de datos relacional.

Problema: Al crear un contacto que está relacionado con un cliente, realiza un bucle de serialización infinita.

- Cliente tiene una lista de Contactos
- Cada Contacto tiene un Cliente
- Que tiene otra vez la lista de Contactos
- Que tienen otra vez el Cliente...

Esto crea una recursión infinita al convertir a JSON, y Spring lanza este error:

Document nesting depth (1001) exceeds the maximum allowed (1000...)

Solución: En Contacto.java añadimos:

```
import com.fasterxml.jackson.annotation.JsonIgnore;
```

```
@ManyToOne
```

```
@JoinColumn(name = "id_cliente", nullable = false)
```

```
@JsonIgnore
```

```
private Cliente cliente;
```

Haber probado con éxito todo el flujo desde usuario hasta actividad significa que ya tenemos una base sólida funcionando:

- Lógica de negocio
- Base de datos
- Seguridad
- Relaciones entre entidades
- API REST lista para conectarse a Flutter

22/04/2025 — EXPANSIÓN DEL BACKEND CON MÓDULOS COMPLETOS Y PRUEBAS

AMPLIACIÓN DE FUNCIONALIDADES

Hoy se han implementado e integrado completamente los siguientes módulos funcionales adicionales, ampliando de forma significativa la capacidad del backend:

MÓDULO DE MATERIALES

Gestionar inventario y descontar automáticamente materiales desde actividades.

- Crear entidad Material con nombre, cantidad, unidad_medida, tipo_material, stock_minimo.
- Crear DTO (MaterialRequest, MaterialResponse), repositorio, servicio y controlador.
- Añadir lógica para restar automáticamente los materiales desde una Actividad.
- Añadir endpoint para ver materiales bajo stock mínimo.

MÓDULO ACTIVIDAD-MATERIAL

- Entidad `ActividadMaterial` para registrar materiales usados por actividad.
- Relación N:M entre Actividad y Material.
- Descuento automático de stock al registrar uso.
- Endpoints: registrar, consultar por actividad, consultar por material.

MÓDULO DE HISTORIAL DE COMPRAS

- Entidad `HistorialCompra` con relación a `Material`.
- Registro de proveedor, fecha, precio unitario, cantidad y descripción.
- Aumento automático del stock al registrar una compra.
- Filtros implementados: por material, por fecha y por rango de fechas.
- Endpoint general para listar todas las compras.

MÓDULO DE ALERTAS

- Entidad `Alerta` relacionada opcionalmente con `Material` o `EventoCalendario`.
- Tipos de alerta: stock, evento (avería opcional)
- Filtros: por tipo, material, evento y fecha.
- Endpoints para ver alertas próximas (posteriores a hoy) y vencidas (anteriores a hoy).

MÓDULO DE EVENTOS DEL CALENDARIO

- Entidad `EventoCalendario` con relación opcional a `Proyecto`.
- Campos: título, descripción, fecha, tipoEvento.
- Endpoints implementados: registrar, listar todos, buscar por tipo, proyecto, fecha exacta, entre fechas, eventos próximos y vencidos.
- Generar alertas (en app o por notificación push más adelante).

PLAN DE PRUEBAS EXHAUSTIVO

- Se diseñó y ejecutó un plan de pruebas paso a paso desde Postman.
- Se verificó la funcionalidad de cada módulo y relación cruzada entre entidades.
- Confirmación de actualización de stock, respuestas esperadas y estados de error gestionados correctamente.

1 MATERIALES

Endpoint base: /api/materiales

1.1 Registrar un nuevo material (POST)

POST /api/materiales

1.2 Listar todos los materiales (GET)

GET /api/materiales

1.3 Buscar por nombre parcial

GET /api/materiales/buscar/nombre?nombre=placa

1.4 Buscar por prefijo ID

GET /api/materiales/buscar/id?prefijo=SCH

1.5 Ver materiales con stock bajo

GET /api/materiales/stock/bajo

1.6 Obtener por ID específico

GET /api/materiales/SCH13138

1.7 Editar material (PUT)

PUT /api/materiales/SCH13138

1.8 Eliminar material (DELETE)

DELETE /api/materiales/SCH13138

2 ACTIVIDAD-MATERIAL

Endpoint base: /api/actividad-material

2.1 Registrar uso de un material en una actividad

POST /api/actividad-material

2.2 Buscar materiales usados en una actividad

GET /api/actividad-material/actividad/1

2.3 Buscar actividades en las que se ha usado un material

GET /api/actividad-material/material/SCH13138

3 HISTORIAL DE COMPRAS

Endpoint base: /api/compras

3.1 Registrar nueva compra

POST /api/compras

3.2 Ver compras por material

GET /api/compras/material/SCH13138

3.3 Ver compras entre fechas

GET /api/compras/fecha?desde=2025-04-01&hasta=2025-04-30

3.4 Listar todas las compras

GET /api/compras

4 ALERTAS

Endpoint base: /api/alertas

4.1 Registrar nueva alerta

POST /api/alertas

4.2 Listar todas las alertas

GET /api/alertas

4.3 Buscar alertas por tipo

GET /api/alertas/tipo/stock

GET /api/alertas/tipo/evento

4.4 Buscar alertas por ID de material

GET /api/alertas/material/SCH13138

4.5 Buscar alertas por ID de evento

GET /api/alertas/evento/1

4.6 Buscar alertas por fecha exacta

GET /api/alertas/fecha?fecha=2025-05-10

4.7 Ver alertas próximas

GET /api/alertas/proximas

4.8 Ver alertas vencidas

GET /api/alertas/vencidas

5 EVENTO CALENDARIO

Endpoint base: /api/eventos

5.1 Registrar nuevo evento

POST /api/eventos

5.2 Listar todos los eventos

GET /api/eventos

5.3 Buscar eventos por tipo

GET /api/eventos/tipo/ITV

GET /api/eventos/tipo/mantenimiento

GET /api/eventos/tipo/revisión

5.4 Buscar eventos por proyecto

GET /api/eventos/proyecto/1

5.5 Buscar eventos por fecha exacta

GET /api/eventos/fecha?fecha=2025-06-01

5.6 Buscar eventos entre dos fechas

GET /api/eventos/entre-fechas?desde=2025-05-01&hasta=2025-06-30

5.7 Ver eventos próximos

GET /api/eventos/proximos

5.8 Ver eventos vencidos

GET /api/eventos/vencidos

RESULTADO

Después de los errores obtenidos ayer, hoy todos los módulos funcionan correctamente y están listos para integrarse con el frontend Flutter. La base del sistema ya permite trazabilidad completa de actividades, materiales, compras y eventos futuros, además de generar alertas dinámicas para la gestión preventiva.

23/04/2025 — EXPANSIÓN DE FUNCIONALIDADES Y CIERRE DEL MÓDULO DE ALBARANES

Lo primero creamos un repositorio de github y subimos el progreso hasta el día de ayer, para llevar el contro de versiones, ya que sabemos que lo anterior funciona correctamente.

MÓDULO DE ALBARANES – COMPLETADO

- Se definió la entidad `Albaran` con ID manual tipo `AVI25059`, siguiendo ejemplos reales de Indasor 66.

- Campos: idAlbaran, actividad, fechaGeneracion, estado, rutaPdf, totalCalculado.

- Se implementó el repositorio y el DTO `AlbaranRequest` para registro de albaranes.
- Controlador con endpoints para:
 - Registrar un albarán.
 - Consultar por ID y por actividad.
 - Listar todos los albaranes registrados.
 - Eliminar un albarán por ID.

Pruebas realizadas en Postman y verificadas en MySQL Workbench:

- Al registrar un albarán, el sistema valida si ya existe uno para la misma actividad (relación 1:1).
- El campo `totalCalculado` ya no se introduce desde el frontend, sino que:
 - Se calcula automáticamente en el backend.
 - Usa una simulación con tarifas fijas:
 - 25€/hora trabajada (calculadas desde `horaEntrada` y `horaSalida` en la actividad).
 - 5€ por unidad de material registrado en `ActividadMaterial`.
 - Resultado redondeado con precisión decimal (2 cifras).
- Se añadió además lógica para simular la generación de PDF:
 - Al registrar un albarán, el sistema genera automáticamente una ruta en formato `albaranes/<built-in function id>.pdf`.
 - Esta ruta se guarda en `rutaPdf` aunque el archivo no se genere aún.
 - Pensado para facilitar el consumo por el frontend y futuras integraciones de exportación.

ACTUALIZACIÓN DEL ESTADO DEL ALBARÁN

- Nuevo endpoint para modificar el estado (`pendiente`, `aprobado`, `enviado`) desde el backend.

- DTO `AlbaranEstadoRequest`.
- Método en el servicio que actualiza solo el campo `estado`, sin afectar el resto del objeto.
- Endpoint probado correctamente en Postman.

FUNCIONALIDAD AÑADIDA AL MÓDULO DE PROYECTOS

- Se replicó la funcionalidad de actualización de estado para la entidad `Proyecto`.
- DTO `ProyectoEstadoRequest`.
- Endpoint `PUT /api/proyectos/<built-in function id>/estado` para modificar estados como `En curso`, `Finalizado`, `Cancelado`, etc.
- Probado en Postman y funcional.

Problema: En albaranes realizamos una suma automática de sus valores, pero no devuelve dicho valor total.

Solución: En Actividad.java no está añadida la relación con ActividadMaterial, añadimos:

```
@OneToMany(mappedBy = "actividad", cascade = CascadeType.ALL)
```

```
@JsonIgnore
```

```
private List<ActividadMaterial> materialesUtilizados;
```

y ya podemos obtener un getMaterialesUtilizados para un correcto flujo y ya obtenemos un totalCalculado automáticamente.

CONCLUSIÓN

- Con estos avances, el backend ya está completamente preparado para el manejo de albaranes profesionales con lógica realista.
- La trazabilidad de actividades, materiales usados, compras y generación de documentos está cubierta en su totalidad.
- Las entidades principales (`Actividad`, `Proyecto`, `Albaran`) ya incluyen capacidad de gestión de estados, lo que permite reflejar el progreso operativo de cada fase en la empresa.

Todo probado exhaustivamente con un plan de pruebas como el de ayer en Postman y confirmado en base de datos MySQL.

24/04/2025 — INTEGRACIÓN DEL LOGIN COMPLETO ENTRE BACKEND Y FRONTEND EN INNOVA66

Permitir que el usuario inicie sesión desde Flutter y acceda a su panel según el rol

CONFIGURACIÓN INICIAL DEL FRONTEND (FLUTTER)

- Se creó estructura modular: `lib/data/models`, `repositories`, `services`, `providers`, `screens`, `widgets`.
- Se implementó `ApiService` usando Dio para conexión REST.
- Se creó el modelo `Usuario.dart` con los campos exactos del backend.

CONEXIÓN CON EL BACKEND SPRING BOOT

- Se implementó `UsuarioRepository` que conecta con `/api/usuarios/login`.
- Se corrigió el backend para que el endpoint de login devolviera un objeto `Usuario` en JSON, antes devolvía un booleano y no correspondía con lo que espera Flutter, antes devolvía `ResponseEntity.ok("Login correcto");` como texto plano, lo que generaba errores en Flutter al intentar deserializarlo.
- En backend el login lo realiza correctamente pero el Flutter nos sigue dando fallo al iniciar sesión, se corrigió error por no importar `SelectorPantalla` en `LoginScreen` y no dirigimos a ningún sitio.
- Se reemplazó el retorno en el controlador para que use:
`ResponseEntity<Usuario>`

GESTIÓN DE AUTENTICACIÓN EN FLUTTER

- Se creó `UsuarioProvider` que guarda el objeto `usuarioLogueado` y notifica a la UI.
- Se conectó el login desde `LoginScreen` y se implementó lógica de navegación según rol.
- Se desarrolló el widget `SelectorPantalla` que redirige a `HomeTecnico` o `HomeAdmin` según el `rol`.
- Se corrigió un error importante donde el login no redirigía: faltaba hacer `Navigator.pushReplacement(...)` tras éxito.

VALIDACIÓN COMPLETA

- Se registró un usuario desde Postman y se probó el flujo completo desde Flutter:
 - Formulario de login
 - Conexión al backend
 - Validación de credenciales
 - Redirección al panel correspondiente

GESTIÓN DE REPOSITORIO

- Se organizó correctamente el repositorio Git, subía logs que son innecesarios.
- Se excluyó `/logs/` del control de versiones y se limpió con `git rm --cached`.

CONCLUSIÓN

Se ha logrado un flujo de login realista y profesional totalmente funcional, conectando Flutter con Spring Boot, y gestionando la sesión del usuario según su rol. Este hito marca el inicio de las pantallas dinámicas por perfil, pero nos damos cuenta que vamos más lento al conectar Flutter con backend.

25/04/2025 — IMPLEMENTACIÓN DEL REGISTRO DE JORNADAS CONFORME A NORMATIVA ESPAÑOLA EN INNOVA66

Tras hablar con CEO y observar que estamos registrando inicio y fin de jornada, hemos decidido que nuestra app se válida para el registro de las mismas bajo la normativa que se exige a las empresas, por lo tanto procedemos a actualizar el backend para cumplir la normativa española de registro de jornada laboral (RD-Ley 8/2019 y LOPDGDD), debiendo modificar memoria de requisitos funcionales y avanzar en la estructura del frontend para técnicos y administradores.

ACTUALIZACIÓN DEL BACKEND (SPRING BOOT)

- Se añadió el campo `jornadaCerrada` en `Jornada.java` para marcar la finalización de jornada.
- Se actualizaron getters y setters.

MODIFICACIONES EN `JornadaRepository`

- Se crearon los métodos `findByUsuarioAndJornadaCerradaFalse` y `findByUsuarioAndFechaAndJornadaCerradaFalse`.
- Permiten validar si un técnico tiene una jornada abierta antes de crear otra.

MODIFICACIONES EN `JornadaService`

- `crearJornada` ahora valida que no exista una jornada abierta en el día.
- `finalizarJornada` actualiza `horaFin`, `comio`, `observaciones` y marca `jornadaCerrada = true`.
- Se añadió `existeJornadaAbierta` para futuras consultas de control.

MODIFICACIONES EN `JornadaController`

- Se reforzó el `POST /api/jornadas` para impedir duplicidades.
- Se creó `PUT /api/jornadas/finalizar/{id}` para cierre de jornada seguro.

AVANCE EN EL FRONTEND (FLUTTER)

- Se creó estructura base en `lib/screens` para HomeAdminScreen, HomeTecnicoScreen y pantallas secundarias.

TÉCNICO:

1. Registro de Jornada

Vamos a añadir una pantalla específica para registrar:

- Inicio de jornada (hora automática o seleccionable).
- Fin de jornada (igual).
- ¿Paró a comer? (Checkbox).
- Horas normales y cálculo de horas extra (si supera 8h descontando 1h si comió).

2. Consulta de horas trabajadas

El técnico verá un resumen de sus horas trabajadas:

- Hoy / Esta semana / Este mes.
- También podrá ver histórico mensual.

3. Selección rápida de clientes y materiales

Importantísimo: para facilitar la selección en listas largas:

- Autocomplete TextField (tipo buscador dinámico).
- Conforme escribe parte del nombre o ID, aparecen coincidencias debajo en lista.
- Selecciona uno de los resultados y lo asigna.

4. Drawer en Técnico

Implementar Drawer lateral (menú deslizable) en técnico:

- Inicio
- Registrar Jornada
- Registrar Actividad
- Consultar Actividades
- Consultar Horas Trabajadas

- Consultar Materiales
- Ver Calendario
- Cerrar Sesión

ADMINISTRADOR:

1. Gestión de Técnicos

Añadir sección Usuarios:

- Consultar lista de técnicos.
- Editar o eliminar usuarios.
- Crear nuevo técnico o administrador.

2. Filtros en Inventario

En la lista de materiales:

- Filtro por nombre y tipoMaterial.
- Barra de búsqueda rápida en la parte superior de la lista.

3. Listado de Clientes y Contactos

Pantalla Clientes:

- Ver datos del cliente.
- Añadir/modificar contactos.
- Ver proyectos asociados.

4. Eventos próximos en lateral

Lo vamos a implementar en la HomeAdminScreen:

- Sidebar lateral derecho (ocultable).
- Eventos próximos agrupados: semana actual, mes actual, etc.
- Cada evento:
 - Título
 - Fecha
 - Proyecto/cliente relacionado (si aplica).
 - Tipo de evento (inspección, revisión, stock bajo...).

5. Registro de compras (HistorialCompra)

- Al añadir nueva compra:
 - Proveedor
 - Fecha
 - Precio unitario
 - Cantidad
 - Descripción

4. Drawer en Técnico

- Inicio
 - Inventario
 - Proyectos
 - Clientes
 - Albaranes
 - Eventos
 - Técnicos (Usuarios)
 - Informes
 - Cerrar Sesión
- Se implementó sistema de navegación con rutas nombradas.
 - Se configuró el ``main.dart`` para soportar rutas seguras.
 - Se crearon los Drawer personalizados para técnico y administrador.
 - Se implementó control de sesión con logout limpio en ``UsuarioProvider``, para que usuario desloguee completamente no quede nada en memoria, y asegura que no se pueda volver atrás con el botón de retroceso.
 - Se solucionó el error de "Don't use BuildContext across async gaps" usando ``if (context.mounted)``.

RESOLUCIÓN DE PROBLEMAS

- Decisión de no incluir ``jornadaCerrada`` en el ``JornadaRequest`` para evitar manipulación desde frontend.
- Implementación de validación previa para evitar jornadas abiertas duplicadas.
- Refactorización completa para usar ``Navigator.pushNamed`` con rutas centralizadas.

GESTIÓN DE REPOSITORIO

- Actualizaciones aplicadas localmente.

CONCLUSIÓN

Se ha consolidado un backend robusto que asegura el registro de jornada conforme a los requisitos legales en España, y se ha estructurado el frontend para una navegación segura, escalable y preparada para la gestión de jornadas laborales.

26/04/2025 — FINALIZACIÓN Y PRUEBA DEL REGISTRO DE JORNADAS EN INNOVA66

Terminar el flujo de registro de jornada en Flutter, enlazado con backend Spring Boot, asegurando el inicio y cierre de jornadas conforme a la normativa laboral vigente.

ACTUALIZACIÓN Y DEPURACIÓN DEL BACKEND Y FRONTEND

- Se detectó que el backend devolvía la entidad Jornada incompleta debido al uso de `@JsonIgnore` en la relación con Usuario, lo que impedía que Flutter accediera a los datos necesarios, pero necesario para evitar bucle infinito.
- Para resolverlo, se creó un DTO llamado `JornadaResponse` que contiene sólo los datos relevantes para el frontend.
- Se modificó el `JornadaController` para devolver `JornadaResponse` al crear una jornada, en lugar de la entidad JPA original.

AJUSTES EN FLUTTER

- En `Jornada.dart` se encontró que Flutter intentaba acceder a `usuario.id_usuario`, pero el nuevo backend devuelve `idUsuario` directamente. Se modificó el `fromJson` para leer `id` como `idJornada` y `idUsuario` directamente.
- Se identificó que campos como `horaFin` y `observacionesGenerales` podían venir como `null`, y eso causaba errores de tipo. Se corrigieron declarándolos como `String?`.

- Se implementó un log con `print(response.data)` en el `ApiService` para inspeccionar el JSON recibido, lo cual permitió detectar con precisión los errores de mapeo y solucionarlos.

FLUJO DE JORNADA FUNCIONAL

- Tras realizar los ajustes, se completó la prueba de registro de jornada desde Flutter.
- La jornada se creó correctamente en la base de datos (MySQL) y se recibió el JSON esperado en la app.
- La jornada fue cargada correctamente en `JornadaProvider` y el sistema mostró el mensaje de éxito mediante `SnackBar`.

AJUSTE DE NAVEGACIÓN EN FLUTTER

- Se observó que al volver desde `JornadaScreen`, Flutter mostraba una advertencia de "¡Atención! Tienes una jornada activa sin finalizar" y no podíamos desplazarnos a otra pantalla.
- Aunque este control era correcto en otros contextos (cerrar sesión o app), no era necesario al volver al Home.
- Se ajustó el `WillPopScope` para permitir salir de la pantalla sin advertencia.

CONTROL DE SEGURIDAD MANTENIDO EN Drawer

- Se mantuvo la advertencia de "Tienes una jornada activa sin finalizar" al intentar cerrar sesión desde el Drawer, lo cual garantiza que el técnico no pueda cerrar sesión sin fichar primero.

CONCLUSIÓN

El sistema de registro de jornada diario ha quedado completamente funcional en Innova66. Los técnicos pueden iniciar jornada, navegar por la app y mantenerla activa correctamente. Todos los problemas detectados de serialización, estructura JSON y comportamiento inesperado de la app han sido solucionados.

27/04/2025 — GESTIÓN DE JORNADA ACTIVA Y RECUPERACIÓN TRAS CIERRE EN INNOVA66

Resolver el escenario en el que la aplicación se cierra inesperadamente (por batería, reinicio o cierre manual) y el técnico queda con una jornada iniciada pero sin datos cargados en Flutter.

IMPLEMENTACIÓN Y PRUEBAS DE CARGA DE JORNADA ACTIVA

- Se identificó que tras reiniciar el dispositivo, Flutter perdía la referencia a la jornada iniciada porque esta solo se mantenía en memoria (JornadaProvider).
- Se propuso la solución de consultar al backend tras login para recuperar cualquier jornada activa pendiente del técnico.
- Se diseñó y documentó un endpoint: GET /api/jornadas/usuario/{id}/activa, encargado de devolver una jornada activa en curso para ese usuario, si existe.
- Se programó la lógica para que Flutter al iniciar sesión, consulte automáticamente este endpoint y, en caso de recibir una jornada, la cargue en JornadaProvider como si nunca se hubiese perdido.

PRUEBA DE RESTAURACIÓN

- Se simuló el cierre completo de la app y posterior reinicio.
- Al iniciar sesión nuevamente, Flutter detectó que el backend devolvía una jornada activa.
- Se cargó correctamente en JornadaProvider y se reestableció el estado de la sesión.
- El técnico pudo continuar su trabajo normal desde el Home, sin restricciones.

CONCLUSIÓN

La gestión de continuidad de jornada tras cierre inesperado ha quedado completamente resuelta. El sistema ahora es capaz de restaurar automáticamente el estado del técnico tras volver a iniciar sesión, permitiendo mantener la trazabilidad y el cumplimiento normativo. Esto cubre uno de los escenarios más críticos en movilidad laboral.

Este módulo ha conllevado más tiempo del planteado por la importancia del mismo.

El siguiente módulo relevante es generar actividades una vez hayamos iniciado jornada, pero para este aspecto han surgido varias dudas, por lo que, debo reunirme con el Indasor para aclararlas, ya que es importante y de ello depende la generación de los albaranes, dudas como en el momento de generar un albarán quien registra los materiales utilizados si hay más de dos técnicos en el mismo proyecto, porque si los dos en su actividad registran un material utilizado, éste se descontará dos veces del inventario, si están dos técnicos en el concepto mano de obra es el tiempo que ha tardado la intervención, independientemente si están dos o más técnicos o es la suma de las horas por cada técnico, entre otras consultas más.

31/04/2025 — MEJORA DE UX RELACIONADO CON JORNADAS Y PREPRACIÓN PANTALLA HOME DE TECNICOS EN INNOVA66

OBJETIVO DEL DÍA: Ahora que ya tenemos un registro de jornada funcional que cumple con la normativa vamos a mejorar UX para que sea más robusta y de esta manera se evite accidentes, como cierres de sesión sin finalizar una jornada.

Cuando el usuario pulse "Cerrar sesión" en el Drawer, si tiene una jornada activa sin finalizar:

- Mostrar advertencia:

"Tienes una jornada activa sin finalizar. ¿Seguro que quieres cerrar sesión?"

- Si confirma: cerramos sesión y limpiamos todo.
- Si cancela: se queda dentro de la app.

En el Drawer cuando tenga iniciada sesión:

- Cambiar el fondo por un color más intenso.
- Cambiar el ícono de usuario por el de un reloj.
- Cambiar de texto "Panel técnico" por "Jornada activa".

En HomeScreen de técnico mostrar un mensaje (en un Card):

"¡Tienes una jornada activa! No olvides finalizar al terminar tu turno.

HOMESCREEN DE TECNICOS

Mostrar menú rápido (botones grandes) para:

- Registrar actividad
- Consultar materiales
- Ver calendario
- Consular mis horas trabajadas

PROBLEMAS DETECTADOS Y RESUELTOS

Al trabajar únicamente con flutter, ya que la conexión con la bases de datos está correctamente implementada y funcionando, no han surgido muchos inconvenientes, salvo los propios del correcto funcionamiento de funciones síncronas o asíncronas, daba fallo en await y líneas como esta:

- final salir = await _mostrarDialogoCerrarSesionConJornada(context);

dicha función sólo se puede usar en funciones marcadas con async.

Fallo en jornadaProvider dentro de este código:

```
const DrawerHeader( decoration: BoxDecoration( color:
jornadaProvider.tieneJornadaAbierta ? Colors.orange : const Color(0xFFFFFB133), ),
```

Solución: no se puede usar const si dentro tenemos variables dinámicas.

CONCLUSIÓN

Hoy hemos reforzado la experiencia de usuario en la aplicación Innova66 para los técnicos, mejorando la robustez y evitando posibles errores de uso durante la jornada laboral. Se implementaron controles visuales y lógicos que advierten al usuario si intenta cerrar sesión con una jornada activa, y se mejoró la interfaz para reflejar de forma clara el estado de la jornada (tanto en el Drawer como en la HomeScreen). Estos cambios no solo mejoran la usabilidad, sino que también aseguran que la trazabilidad del registro de jornada cumpla con la normativa sin riesgo de errores accidentales.

01/05/2025 — DESARROLLO MÓDULOS INVENTARIO, PROYECTOS Y CLIENTES EN INNOVA66

INVENTARIO

Crear toda la parte visual y funcional para que el admin pueda gestionar materiales desde la app:

Lista de materiales:

- Mostrar: nombre, tipo, cantidad disponible, unidad de medida, stock mínimo.
- *Visual:* Añadir un ícono de alerta si la cantidad está por debajo del stockMinimo.

Añadir material:

- Formulario: nombre, descripción, tipo, unidad de medida, cantidad, stock mínimo.

Editar material:

- Permitir edición completa de los datos.

Eliminar material

Para ello creamos las clases .dart:

- material
- material_repository
- inventario_screen

Creación de formulario que lo utilizaremos tanto para un nuevo material como para la edición del mismo, controlando introducción de campos clave como unidad de medida, tipo material, sean siempre de la misma manera porque se utilizarán más adelante para asignar filtros o búsquedas.

En pantalla Inventario_screen implementar un buscador, seleccionable por nombre o por ID, botón para limpiar búsqueda, desplegable para filtrar materiales por tipo y botón para mostrar materiales que tienen stock bajo.

Problemas:

- No carga lista de materiales en pantalla, lanza error Exception has occurred
- No podemos filtrar materiales por tipo.

Solución:

- Como antes no se controlaba la forma de introducir unidad de medida y tipo de material, algún nombre no coincidía por lo que desde MySQL Workbench procedemos a modificar para que esto no suceda.
- Para seleccionar materiales por tipo añadimos en backend nuevo endpoint para conseguirlo (modificar controller, service, repository)

CLIENTES

- Mostrar en una pantalla:
 - Lista completa de Clientes:
 - Nombre, CIF, Dirección.
 - Mostrar también sus Contactos asociados:
 - Nombre del contacto, cargo, teléfono, correo.
- Implementación de CRUD para clientes y contactos.

Dentro de carpeta widgets creamos otra para admin y técnico para situar ahí pantallas como formularios y mantener orden y limpieza.

Para ello creamos las clases .dart:

- cliente
- contacto
- cliente_repository
- clientes_screen

Problemas:

- No carga lista de clientes
- No tenemos todos los endpoints necesarios para CRUD

Soluciones:

- Recibe un atributo null, en flutter una variable primitiva (boolean) no puede ser null, pero si lo cambiamos a tipo de una clase no primitiva (Boolean) resolveremos parte del problema, probando con Postman y recibiendo datos correctamente desde backend y observamos también que nombre del id de cliente no coincide con frontend, procediendo a modificarlo.
- Añadir endpoint necesarios en backend (modificando todas clases afectadas).
- Añadir en la relación cliente-contacto sentencia para que al eliminar un cliente borre en cascada los contactos relacionados a él.

PROYECTOS

- Pantalla de Proyectos:

Mostrar lista de proyectos:

- Nombre, estado, cliente, ubicación, descripción, fecha inicio.
- Botón para crear proyecto.
- Ícono para editar proyecto.
- Ícono para eliminar proyecto.
- Añadir filtro por clientes, cargándose desde la base de datos siempre actualizados.
- Formulario Proyecto
- Añadir función para evitar duplicados de nombres de proyectos para un mismo cliente

Para ello creamos las clases .dart:

- proyecto
- proyecto_repository
- formulario_proyecto
- proyectos_screen

Problemas:

- Para variar no carga en pantalla proyectos.
- Al añadir función para evitar duplicados de clientes en el backend, éste deja de funcionar.

Solución:

- Al añadir JsonIgnore en la relacion de cliente con proyectos, backend no envía los datos del cliente, siendo necesario recibir el id del cliente, en este caso se pudo solucionar con la siguiente sentencia:

```
@ManyToOne @JoinColumn(name = "id_cliente", nullable = false)
@JsonIgnoreProperties({"proyectos", "contactos"}) private Cliente cliente;
```

- Existe problema con la forma de enviar y recibir el id de un cliente a la hora de definir un método para usar relaciones cruzadas, lo tengo marcado como id_cliente y el método necesita que se llame id, cambiar el nombre de esto me implica cambiarlo en varias clases tanto dentro de backend como frontend, por lo que, he encontrado la manera con una consulta de recibir dicho campo con el nombre id sin tener que cambiar el nombre a la variable añadiendo esto:

```
@Query("SELECT COUNT(p) > 0 FROM Proyecto p WHERE p.nombre = :nombre AND  
p.cliente.id_cliente = :idCliente")
```

```
boolean existsByNombreAndCliente(@Param("nombre") String nombre, @Param("idCliente") Long  
idCliente);
```

CONCLUSIÓN

En la jornada de desarrollo se ha completado con éxito la implementación de los módulos Inventario, Clientes y Proyectos dentro de la app Innova66, integrando tanto la parte visual (Flutter) como la lógica funcional completa en el backend (Spring Boot).

Se logró establecer un flujo sólido para la gestión de materiales, clientes (junto a sus contactos) y proyectos, implementando CRUDs completos y dotando a las pantallas de funcionalidades avanzadas como filtros dinámicos, búsqueda por distintos criterios y validaciones estrictas para evitar inconsistencias (por ejemplo, duplicados en proyectos).

La arquitectura quedó robusta y alineada, permitiendo una gestión completa de inventario, clientes y proyectos desde la app, y asegurando una experiencia fluida y fiable para los usuarios administradores. El proyecto avanza en muy buen ritmo, resolviendo con éxito problemas complejos y afinando detalles críticos en cada iteración.

02/05/2025 — Desarrollo de módulo Albaranes con filtros avanzados y refactorización DTO

- Solucionar la falta de información de Proyecto y Cliente en los albaranes para poder implementar filtros avanzados en el frontend.
- Evitar bucles infinitos en la serialización JSON de entidades relacionadas.
- Refactorizar la respuesta de la API /api/albaranes para devolver los datos de forma plana mediante un DTO.
- Implementar en el frontend los filtros por Cliente y Proyecto, además de mantener búsqueda por ID y filtro por Estado.

CAMBIOS REALIZADOS EN BACKEND (SPRING BOOT)

Problemas:

- Al intentar exponer la relación Actividad -> Proyecto -> Cliente en el JSON de albaranes, surgieron bucles infinitos de serialización (Actividad incluía Proyecto, que incluía la lista de Actividades de nuevo, etc.).

Solución:

- Se creó un DTO específico AlbaranResponse para devolver la información de forma plana, evitando exponer directamente las entidades JPA.
- Se modificó el endpoint GET /api/albaranes para devolver una lista de AlbaranResponse en lugar de la entidad Albaran completa.
- Se creó un mapeo manual en el controller para transformar cada Albaran en su DTO correspondiente, evitando relaciones cruzadas.

CAMBIOS REALIZADOS EN FRONTEND (FLUTTER)

Adaptación del modelo Albaran:

- Se modificó el modelo Albaran para incluir los nuevos campos planos proporcionados por el backend.
- Se eliminó la estructura anterior que contenía objetos anidados (actividad, proyecto, cliente) para simplificar la deserialización.

Implementación de filtros avanzados:

- Se añadió un filtro por Cliente (Dropdown), que carga dinámicamente la lista de clientes desde la API /api/clientes.
- Se implementó un filtro por Proyecto (Dropdown), que se actualiza dinámicamente según el Cliente seleccionado.
- Se implementó un filtro por EStado (pendiente, aprobado, enviado)
- Se implementó un buscador por ID.

Pantalla AlbaranesScreen:

- Se actualizó la UI para incluir los nuevos filtros y gestionar la carga dinámica de Clientes y Proyectos.
- Se agregó lógica para evitar que el filtro de Proyectos muestre valores antes de seleccionar Cliente.

Refactorización de ClienteRepository:

- Se reorganizó para incluir dos métodos separados:
 - fetchAll() → devuelve la lista completa de Clientes (para gestión).
 - fetchResumidos() → devuelve un resumen (id + nombre) para los filtros.

Tarjeta del albarán:

- Se muestra:
 - Proyecto asociado.
 - Cliente asociado.
 - Estado y Total, manteniendo la estética limpia y clara.

Resolución de errores:

- Se corrigió un error de conversión en cliente_resumen.dart donde id_cliente era numérico y se esperaba un String → Solución: .toString() en la conversión.
- Se ajustaron las instancias de ClienteRepository para asegurar que recibían correctamente la dependencia ApiService.

RESULTADO FINAL

La pantalla de gestión de Albaranes ahora ofrece:

- Filtros avanzados por Cliente, Proyecto, Estado e ID.
- Carga dinámica y actualizada de la lista de Clientes y Proyectos desde el backend.

- Visualización limpia y detallada en las tarjetas y en el detalle de cada Albarán.
- Botón para ver Albarán en PDF (aún no implementado)

CONCLUSIÓN

Hemos cerrado la ampliación del módulo de Albaranes dejando preparada la base para futuras ampliaciones (como búsquedas más avanzadas o exportación de PDFs reales). La arquitectura backend se mantiene limpia gracias a la introducción de DTOs y el frontend ofrece una experiencia completa y profesional para la gestión.

03-04/05/2025 MÓDULO CRUD USUARIOS

1. Implementación completa del módulo de Usuarios (CRUD completo)

Nos permite:

- Registrar usuarios.
- Listarlos y filtrarlos en tiempo real (por nombre/email).
- Editarlos.
- Activarlos/desactivarlos en lugar de eliminarlos (cumpliendo la normativa de conservación de datos).

En backend:

- Añadimos el campo activo en la base de datos y en la clase Usuario.
- Endpoints creados:
 - POST para registrar.
 - PUT para actualizar.
 - PUT /activar y /desactivar para activar/desactivar usuarios.
 - GET para listar todos (con filtrado por activos/desactivados).
 - DELETE mantenido solo para pruebas (pero desaconsejado para producción).
- Se probó todo en Postman.

En flutter:

- Pantalla TecnicosScreen:
 - Buscador en tiempo real (nombre/email).
 - Listado separado en dos modos: activos y desactivados (botón de alternar).
 - Botones para activar/desactivar usuarios (eliminar ha sido retirado).
 - Modal para confirmaciones de desactivación.
- Pantalla FormularioUsuario:
 - Registro y edición de usuarios.
 - Listas desplegables para roles (sin permitir escritura manual).

Problema detectado durante la implementación:

- Login bloqueado para usuarios nuevos/actualizados: Al intentar iniciar sesión con usuarios nuevos o actualizados, no se cargaba la pantalla HomeTecnico tras el login. Aunque en la base de datos los datos eran correctos y las respuestas del backend eran 200 OK, Flutter no redirigía como debía.
- Diagnóstico: Revisando los logs se identificó que los usuarios recién creados o actualizados enviaban "rol": "tecnico" (sin acento), mientras que los usuarios antiguos tenían "rol": "técnico" (con acento). En Flutter, el if (rol == 'técnico') no coincidía con 'tecnico'.
- Solución aplicada: Se confirmó que el DropdownButtonFormField para seleccionar el rol del usuario en el formulario solo ofrecía valores definidos (tecnico y administrador). La clave era asegurar que backend y frontend fueran coherentes:
 - Se revisaron y sincronizaron los valores de rol.
 - Se realizó un cambio en Flutter para aceptar ambos formatos (añadiendo .toLowerCase().contains('tecnico') temporalmente).
 - También se reforzó en backend la consistencia de los datos creados.

Resultado: El problema quedó solucionado y todos los usuarios, tanto nuevos como antiguos, pueden iniciar sesión correctamente.

Problema con la eliminación de usuarios:

- Situación inicial: Intentar eliminar un usuario que tuviera actividades o jornadas daba error de integridad referencial

(SQLIntegrityConstraintViolationException). Esto es lógico porque las tablas estaban relacionadas.

- Decisión: Se decidió implementar una estrategia de desactivación de usuario en lugar de eliminación física para cumplir con requisitos legales (almacenar datos al menos 4 años).

Cambios aplicados:

- Base de datos: Se añadió un campo activo (boolean) en la tabla usuario.
- Modelo Usuario: Se añadió el campo activo con sus getters/setters.
- Controller:
 - Nuevo endpoint PUT /api/usuarios/{id}/desactivar para desactivar.
 - Nuevo endpoint PUT /api/usuarios/{id}/activar para reactivar.
- Repository Flutter: Se añadieron métodos activarUsuario y desactivarUsuario.
- Pantalla TecnicosScreen:
 - Se eliminaron los botones para borrar usuarios.
 - Se añadieron opciones para activar/desactivar usuarios.
 - Se añadió un botón para alternar entre la lista de usuarios activos y desactivados.

Pruebas: Todo fue probado tanto desde Flutter como desde Postman, funcionando correctamente.

2. Mejora UX en Inventario: búsqueda más fluida

Situación inicial: La búsqueda en la pantalla de inventario requería elegir entre "nombre" o "ID", escribir la búsqueda y luego pulsar en la lupa o Enter.

Cambio aplicado:

- Se eliminó el selector de tipo de búsqueda (nombre o ID).
- Se mejoró la UX para hacerla como en la TecnicosScreen: ahora el buscador es reactivo y filtra en tiempo real conforme se escribe texto (por nombre o ID a la vez).
- Se mantuvieron los filtros por tipo de material y stock bajo, y el botón de limpiar búsqueda sigue disponible.

3. Implementación completa del módulo Historial de Compras

Backend:

- Modelo HistorialCompra: Confirmado y validado.
- Repository: Añadidas queries adicionales para filtrar por proveedor y fechas.
- Service: Se añadió la lógica para:
 - Registrar compras (incrementando el stock automáticamente).
 - Filtrar por material, proveedor, fechas, etc.
- Controller: Se añadieron todos los endpoints necesarios, y se migró a HistorialCompraResponse para mejorar la respuesta.

Flutter:

- Modelo HistorialCompra: Creado para reflejar la respuesta del backend.
- Repository: Métodos para registrar y consultar compras.
- Pantalla HistorialCompraScreen:
 - Lista completa de compras (con recarga y buscador).
 - Añadido un formulario mejorado para registrar compras.
 - Implementamos un RawAutocomplete para seleccionar el material de la compra, lo que mejora notablemente la UX si hay muchos materiales (comparado con los DropdownButton anteriores).

Pruebas: Todos los endpoints verificados desde Postman y toda la funcionalidad en Flutter validada.

4. Implementación completa del módulo Eventos de Calendario

Backend:

- Modelo EventoCalendario: Confirmado.
- Repository: Añadidas queries para filtrar por proyecto, tipo, fechas, eventos próximos/vencidos.
- Service: Lógica para registrar, actualizar y buscar eventos.
- Controller: Endpoints completos, usando EventoCalendarioResponse.

Flutter:

- Modelo EventoCalendario: Creado y probado.

- Repository: Métodos para todas las operaciones CRUD y filtros.
- Pantalla EventosCalendarioScreen:
 - Buscador mejorado con filtrado en tiempo real.
 - Filtros rápidos: *próximos* y *vencidos*.
 - Botón para *resetear la lista completa* tras aplicar filtros (ícono de lista).
 - Añadido un formulario para registrar o actualizar eventos.
 - La selección de proyecto se hizo opcional y se añadió la opción "Ninguno" para poder limpiar la selección si se elige por error.
- Mejoras UX adicionales:
 - Añadido el botón para limpiar búsqueda en el TextField.
 - Añadido un botón específico para *mostrar todos los eventos* tras aplicar filtros (ícono de lista).

Otros errores solucionados:

Flutter: Un error al hacer setState después del dispose en el login quedó revisado tras resolver el problema inicial del rol.

Backend: Ajustes en la serialización de respuestas para asegurar compatibilidad total.

Conclusión

Ha sido una jornada muy productiva y completa. Se resolvieron problemas críticos en el login, se implementó un CRUD completo de Usuarios, y se desarrollaron dos módulos completos (Historial de Compras y Eventos de Calendario) tanto en backend como en frontend. Además, se mejoró la experiencia de usuario en pantallas clave como Inventario y Eventos, y se dejó todo probado y funcionando.

06-07/05/2025 — MEJORA DE MÓDULO DE ALERTAS Y CALENDARIO, Y AJUSTES INVENTARIO EN INNOVA66

ALERTAS DE STOCK Y EVENTOS

Implementar alertas automáticas de stock bajo y eventos próximos que se visualicen en el *HomeAdminScreen* de manera dinámica y coherente.

Backend (Spring Boot):

- Modificamos AlertaRepository para incluir:
 - findByTipoAlertaAndMaterialAndEstado
 - findByTipoAlertaAndMaterialIdAndEstado
- Actualizamos AlertaService:
 - Creamos cerrarAlertaStockSiExiste para cambiar el estado de la alerta a *resuelta* cuando un material se repone.
 - Reforzamos la lógica en crearAlertaStockSiNoExiste para:
 - Si existe alerta en estado *resuelta*, se reactiva.
 - Si no existe, se crea una nueva.
- En MaterialService:
 - Modificamos actualizarMaterial para:
 - Verificar el stock tras la actualización.
 - Crear o reactivar alerta si el stock queda por debajo del mínimo.
 - Cerrar la alerta si el stock vuelve a la normalidad.
- Creamos un nuevo endpoint en AlertaController para devolver únicamente alertas activas por tipo (optimización para frontend).

Problemas detectados y resueltos:

- Error en AlertaRepository por uso incorrecto del nombre de atributo (IdMaterial en lugar de MaterialId).
- Ajustamos el nombre y la lógica para mapear correctamente material.id en la query.

MEJORA DE CALENDARIO Y EVENTOS (Flutter)

- En EventosCalendarioScreen rediseñamos la pantalla para usar TableCalendar:
 - Muestra los eventos agrupados por fechas.
 - Al hacer clic en un día se muestran los eventos de ese día.
 - UX refinada con colores según tipo de evento y agrupación visual.

- Integramos un buscador lateral que permite filtrar eventos por título o proyecto.
- Añadimos un Dropdown para ajustar el rango de días (7, 15, 30) en el panel lateral derecho de *HomeAdminScreen* para eventos próximos.
- Creamos *AlertaProvider* y *AlertaRepository* en Flutter para gestionar las alertas (stock, eventos y futuras alertas SMS).
- Problemas UX:
 - Solucionamos errores de *notifyListeners* que se lanzaban durante el build añadiendo la carga de datos en *addPostFrameCallback*.
 - Mejoramos la agrupación en *TableCalendar* para evitar duplicados de fechas.
 - Ajustamos la consulta para que el panel lateral solo mostrara las alertas con estado *activa* y evitar que aparecieran alertas resueltas.

MODIFICACIONES EN INVENTARIO (Flutter)

En *InventarioScreen* revisamos la lógica del filtro "Mostrar materiales con stock bajo" para que funcione en sincronía con el backend y la nueva lógica de alertas.

Probamos la actualización del material en *FormularioMaterial* asegurando:

- Que al bajar la cantidad por debajo del mínimo se genera la alerta.
- Que al reponer se actualiza a *resuelta*.
- El icono de la lista cambia dinámicamente mostrando advertencia solo cuando corresponde.

CONCLUSIÓN

Han sido dos jornadas intensas pero muy productivas:

- La lógica de alertas ha quedado robusta y testeada desde backend y frontend.
- El calendario ha mejorado en funcionalidad y UX.
- Hemos solucionado problemas críticos relacionados con duplicidad de alertas y refresco en pantalla.
- Dejamos preparado el entorno para afrontar la generación de PDF en los albaranes (pendiente para mañana).

09/05/2025 — Generación de PDF profesional para Albaranes en INNOVA66

Implementar un sistema de generación de albaranes en PDF con diseño profesional, válido para enviar a clientes.

Cambios implementados en backend (Spring Boot)

- Creación de clase PdfGeneratorUtil
 - Generación real de archivos PDF con OpenPDF.
 - Ruta de guardado: src/main/resources/static/pdfs/albaranes/.
 - Se añadió dependencia al pom.xml.
- Modificación de AlbaranService
 - Al registrar un nuevo albarán se genera automáticamente el PDF usando PdfGeneratorUtil.
 - Se guarda la ruta real del archivo en el campo rutaPdf.
- Nuevo endpoint en AlbaranController
 - GET /api/albaranes/{id}/pdf para descargar el PDF generado.
 - Devuelve el archivo como application/pdf con Content-Disposition: inline.

Diseño profesional del PDF

Se definió y se implementó el diseño definitivo del albarán, incluyendo:

Cabecera:

- Logo de Indasor 66 (ubicado en src/main/resources/static/logo/Logo_Indasor.png).
- Datos de empresa justo debajo del logo, compactados.

Cliente:

- Alineado a la derecha en celda con borde visible, sin fondo.

Título:

- Centrado, grande: ALBARÁN AVIxxxxx.

Detalle:

- Tabla con Fecha y N° Cliente.

Contenido principal:

- Tabla con columnas: Ref. (material o MO), Descripción, Ud. (cantidad o duración en horas).

- Color de cabecera en naranja corporativo #FE9517.

Pie:

- Firma del cliente (línea).

Se realizaron ajustes finos de márgenes, paddings y espaciado hasta obtener un resultado visual profesional.

Resultados de prueba

Se probaron múltiples albaranes (AVI25062, AVI25063) generando y descargando los PDFs para validación visual.

Se detectó y corrigió un error por uso incorrecto de PdfPCell dentro de otra PdfPCell.

Albarán proporcionado por INDASOR:

		Cooperativa	
C/. Veintitres de abril, 25 50014 ZARAGOZA		Diseminados	
C.I.F. B99459943 administracion@indasor.es		50750 Pina De Ebro ZARAGOZA	
Nº de Albarán AVI/25059	Fecha 06/03/2025	Nº de Cli./Prov. 4300003	CIF :
			Pág. 1
Ref.	Descripción	Ud.	
MO	Mano de obra. Instalación de nuevo elevador entrada secadero de maíz. Pasar cables a motor, cilindro neumático, detector de giro y finales de carrera. Colocar detector de giro y electroválvula para accionamiento cilindro. Colocar filtro y regulador de aire comprimido. Montar sistema antiretorno elevador.	18	
RVK3X2.5	Manguera aceflex 0.6/1Kv. 3 x 2,5 mm	35	
RVK10X1S	Sumiflex® VV-F 10 G 1mm² 300/500V CPR Eca	35	
RVK2X1	Cable flexible 500V de 2 x 1 mm	42	
CAJEST110	Caja estanca 110X110X60	1	
TUBMET16	Tubo metalplas PG 16	6	
TUBMET11	Tubo metalplas PG 11	28	
FILAIR1/2	Filtro aire + aceite 1/2"	1	
TUBPOL10X	Tubo poliamida 10X8	10	
ELEMON5/2	Electroválvula monoestable 5/2 1/8 1 bobina	1	
CONVC230	Controlador de rotación Filsa VC con detector A 230V	1	
VAR	Reducciones, machones, llave paso, bridas y accesorios	1	

Recibí Conforme:

Albarán generado con OpenPDF:



INDASOR 66 S.L.
 CIF: B99459943
 C/ Veintitres de abril 25
 50014 Zaragoza
 Tel: 976 123 456
 administracion@indasor.es

Cliente Solar
 Calle Energía 42 de la madre
 CIF: B12345678

ALBARÁN AVI25063

Fecha: 09/05/2025
Nº Cliente: 100001

Ref.	Descripción	Ud.
MO	Instalación de paneles	4,50
SCH13138	Placa 90X100mm	2

Firma del cliente: _____

11/05/2025 — Reunión con CEO de Indasor

Revisar pantallas implementadas para pulir detalles:

Jornada:

- Totalmente conforme con el método utilizado para el registro de la jornada de los técnicos y UX empleado para evitar que no se cierre una jornada.

Inventario:

- Se definen todos los grupos que engloban a los materiales (automáticos, contactores, disyuntores, cables, vigilancia, motores, bobinados, cajas, mecanismos, iluminación).
- Pantalla y filtros implementados conforme.

Proyectos:

- Añadir filtro por estado "Pendientes", para días en el que se tiene tiempo libre, se remata este tipo de proyectos y cerrarlos del todo.
- Pantalla y demás filtros conforme.

Eventos:

- Se definen 4 grupos de eventos:
 - ITV preaviso de 30 días
 - Mantenimiento (instalaciones, vehículos) preaviso de 30 días
 - Revisiones (alta y baja tensión) preaviso de 30 días
 - Seguros (vehículos, instalaciones) preaviso de 60 días

Compras:

- Añadir al precio de material el descuento aplicado en la compra.
- Añadir filtro por artículos y descripción.
- Se quita filtro por proveedor

Albaranes:

- Se define opción de generar albaranes sólo con cantidades utilizadas y otra con precios incluidos.
- Se define ID de albaranes, por ejemplo, AVI25003:
 - AV (albarán ventas), AC (albarán compras) o AG (albarán gastos)
 - I, de Indasor
 - 25, año en curso
 - 003, número de albarán
- Los albaranes se generan automáticamente para cada actividad con los materiales utilizados.
- Los albaranes se terminan de editar por administrador la cantidad de mano de obra (en función de cuantos técnicos han acudido, el descuento que se le hace al cliente,..) descripción de tareas realizadas más detallado del indicado por los técnicos al registrar la actividad y descuento en los materiales.
- Se define la forma de registrar la actividad para no duplicar uso de materiales si dos técnicos registran una actividad en el mismo proyecto.

13/05/2025 — IMPLEMENTACIÓN DE ACTIVIDADES Y PRUEBAS EN EMULADOR ANDROID EN INNOVA66

DESARROLLO DE LA PANTALLA REGISTRO DE ACTIVIDAD

Se desarrolló la pantalla ActividadScreen para el registro de actividades diarias por parte de los técnicos. Dicha pantalla se compone de:

- Campos para cliente, proyecto (opcional), fecha, hora de entrada y salida, descripción de tareas y observaciones.
- Selección de materiales utilizados, añadiendo múltiples líneas con cantidad empleada.
- Validaciones para garantizar que no se envíen campos vacíos ni horas inconsistentes.

AJUSTES EN BACKEND (Spring Boot)

- Se revisaron y ajustaron los DTOs ActividadRequest, ActividadMaterialRequest, y se reforzó la validación en ActividadController y ActividadMaterialController para el correcto tratamiento de relaciones entre

entidades (Actividad, Material, Proyecto...).

- Se integró la lógica para descontar materiales automáticamente al registrar una actividad.
- Se resolvió un problema con la serialización infinita entre Actividad → Proyecto → Cliente → Actividades añadiendo anotaciones `@JsonIgnore` o `@JsonIgnoreProperties`.

PRIMERA PRUEBA DESDE EMULADOR ANDROID

Decidimos probar toda la funcionalidad técnica (login, jornada, actividad) desde un emulador Android (Pixel 5). Aparecieron numerosos problemas:

1. ERROR EN RENDERIZADO DEL EMULADOR

- Al iniciar el emulador aparecían continuamente errores `E/libEGL: called unimplemented OpenGL ES API`.
- Solución: Crear un nuevo emulador y configurar Graphics: Software en la configuración del dispositivo.

2. ERROR DE CONEXIÓN

- Flutter no lograba conectarse al backend Spring Boot.
- Diagnóstico: localhost en Android no se refiere al host físico, sino al emulador.
- Solución: Cambiar localhost por 10.0.2.2 en ApiService.

3. ERROR EN FORMATO DE HORA

- Al registrar jornada o actividad, Flutter enviaba hora en formato 7:19 PM, provocando errores 400 Bad Request en Spring.
- Solución: Implementar función `formatTimeOfDay()` en Flutter para convertir `TimeOfDay` al formato correcto HH:mm:ss.

4. ERROR 400 AL REGISTRAR JORNADA

- Backend rechazaba la petición incluso con los datos correctos.
- Se revisó todo el flujo desde Flutter → Dio → Backend.
- Se añadió `@JsonFormat(pattern = "HH:mm:ss")` en el DTO `JornadaRequest` y se registró manualmente el `JavaTimeModule` en una clase `@Configuration` (`JacksonConfig.java`) para asegurar el parsing correcto de `LocalTime`.

5. FALLO DE LÓGICA DE NEGOCIO

- Tras todas las correcciones, el error persistía.
- Diagnóstico final: ya existía una jornada para ese técnico en ese día (aunque estuviera cerrada), y el sistema lo trataba como duplicada.
- Solución: El sistema estaba funcionando correctamente. El error era una validación del negocio y no técnico.

Se decidió implementar próximamente un mensaje que informe al usuario cuando ya tiene una jornada (cerrada o no) registrada para ese día, y que no puede crear una nueva.

CONCLUSIÓN

Aunque el proceso ha sido más complejo de lo esperado, ha permitido probar a fondo la robustez del backend y frontend. Gracias a las herramientas de logging, el diagnóstico ha sido certero. El sistema de actividades y jornadas ya puede ser usado desde dispositivos móviles y ha quedado validado su correcto funcionamiento.

14/05/2025 — REGISTRO DE ACTIVIDADES SIN JORNADA EN INNOVA66

OBJETIVO

Permitir a los técnicos registrar actividades incluso si no tienen una jornada activa iniciada. Esta funcionalidad responde a un escenario real transmitido por Indasor, en el que los técnicos deben poder intervenir de urgencia (por ejemplo, ante averías), aun fuera de su jornada diaria. Para ello se ha adaptado tanto la base de datos, como el backend y la interfaz de usuario en Flutter.

CAMBIOS EN BACKEND (SPRING BOOT)

Clase Actividad.java

Se modificó la relación con Jornada:

java

Copiar código

@ManyToOne

@JoinColumn(name = "id_jornada")

@JsonIgnore

private Jornada jornada;

- Se eliminó nullable = false para permitir nulos.

SQL aplicado en MySQL Workbench

sql

Copiar código

```
ALTER TABLE actividades MODIFY COLUMN id_jornada BIGINT NULL;
```

Sin esta modificación, el backend devolvía error SQL Column 'id_jornada' cannot be null.

DTO ActividadRequest.java

- Confirmado que idJornada es Long y no tiene restricciones @NotNull, por lo que acepta null.

Servicio ActividadService.java

- No fue necesario modificar esta clase, ya que la lógica de creación está centralizada en el controlador.

Controlador ActividadController.java

Se modificó el método @PostMapping para hacer opcional el uso de jornada:

java

Copiar código

```
Jornada jornada = null;
```

```
if (request.idJornada() != null) {
    jornada = jornadaService.buscarPorId(request.idJornada())
        .orElseThrow(() -> new RuntimeException("Jornada no encontrada"));
}
```

```
actividad.setJornada(jornada); // puede ser null
```

CAMBIOS EN FRONTEND (FLUTTER)

actividad_screen.dart

- Se añadió un CheckboxListTile visible cuando no hay jornada activa, con la etiqueta:

"Registrar actividad urgente (sin jornada)"


- Se añadió un botón "Abrir formulario" que se habilita solo si se marca el checkbox.
- Se modificó el método `_abrirFormularioActividad()`:
 - Antes usaba `showDialog(...)` → Ahora usa `Navigator.push(...)` con pantalla completa.
 - Se le pasa el parámetro `registroSinJornada` al formulario.

dart

Copiar código

```
final resultado = await Navigator.push(
    context,
    MaterialPageRoute(
        builder: (_) => FormularioActividad(registroSinJornada: _registroSinJornada),
```

),
);

 formulario_actividad.dart

- Se adaptó por completo al estilo del resto de formularios como FormularioMaterial.
- Cambios principales:
 - Se encapsuló en un Scaffold con AppBar.
 - Se eliminó el uso de Dialog → ahora se muestra como una pantalla completa.
 - Se añadió el parámetro registroSinJornada como booleano que define el comportamiento.
- Lógica especial para decidir el idJornada, idUsuario y la fecha:

dart

Copiar código

```
final data = {
  'idJornada': widget.registroSinJornada ? null : jornada?.idJornada,
  'idUsuario': widget.registroSinJornada
    ? Provider.of<UsuarioProvider>(context, listen:
false).usuarioLogueado!.idUsuario
    : jornada!.idUsuario,
  'fecha': (widget.registroSinJornada ? DateTime.now() : jornada!.fecha)
    .toIso8601String()
    .split('T')[0],
  ...
};
```

- La lógica para seleccionar cliente, proyecto, hora de entrada/salida, descripción y añadir materiales se mantuvo completamente funcional.

PROBLEMAS DETECTADOS Y SOLUCIONADOS

1. **Error SQL id_jornada cannot be null**

- Causa: aunque en Java se permitió null, la columna en MySQL seguía siendo NOT NULL.
- Solución: se ejecutó un ALTER TABLE para permitir valores nulos.

2. **El frontend no daba feedback tras registrar actividad sin jornada**

- Causa: no se mostraban actividades si no había jornada activa.
- Solución: se añadió un SnackBar de confirmación al guardar actividad sin jornada, y se mantiene la lógica de recarga únicamente si hay jornada activa.

CONCLUSIÓN

Con esta mejora, el sistema Innova66 permite a los técnicos registrar actividades aunque no hayan iniciado una jornada formal. Esto mejora la flexibilidad operativa de la aplicación en escenarios reales, como averías o visitas urgentes. La solución mantiene trazabilidad y compatibilidad con el modelo actual de generación de albaranes, adaptándose a la operativa de Indasor sin comprometer la integridad de los datos.

17/05/2025 — MEJORAS FUNCIONALES Y DE USABILIDAD EN HISTORIAL DE COMPRAS E INICIO DE PLANIFICACIÓN DE EVENTOS

Módulo: Historial de Compras

Objetivo:

Permitir registrar compras con posibles descuentos aplicados por proveedores, calcular el precio final automáticamente y mejorar la experiencia de búsqueda y visualización de compras en la app.

Backend (Spring Boot)

1. Ampliación de la entidad **HistorialCompra**:

- Se añadieron dos nuevos campos:
 - Double descuento → puede ser null (opcional).
 - Double precioFinal → siempre calculado automáticamente en base al precio unitario y el descuento.
- Se modificó la lógica en **HistorialCompraService** para:
 - $\text{Calcular precioFinal} = \text{precioUnitario} * (1 - \text{descuento} / 100)$, usando 0.0 como valor por defecto si descuento es null.

2. Actualización de DTOs:

- **HistorialCompraRequest**: se añadió descuento.
- **HistorialCompraResponse**: se añadieron descuento y precioFinal, junto con su integración en el método **fromEntity()**.

Frontend (Flutter)

1. Modelo **HistorialCompra**:

- Se añadieron las propiedades descuento y precioFinal.
- Se actualizaron los métodos **fromJson()** y **toJson()**.

2. Formulario de registro (**FormularioHistorialCompra**):

- Se añadió un nuevo campo **TextFormField** para introducir el descuento.
- Se integró en el mapa data que se envía al backend si no está vacío.

3. Pantalla de visualización (HistorialCompraScreen):

- Se actualizó la tarjeta de cada compra para mostrar:
 - El descuento (si existe).
 - El precio final calculado.
- Se hizo un rediseño del buscador:
 - Ahora solo permite buscar por **nombre o ID** de material.
 - Se eliminó el filtro por proveedor desde el buscador directo.
- Se mejoró el botón Icons.sort:
 - Ahora ordena cualquier conjunto visible (filtrado o no) por fecha descendente.
 - Se implementó lógica toggle: si ya está ordenado, vuelve al estado anterior (guardado previamente con List.from()).
 - Se añadió un indicador visual con cambio de color (Colors.orange) cuando está activo.
 - Se solucionó una advertencia de Dart sobre el uso de ?? cuando ya no era necesario usando null-safety explícita.

4. Botón Icons.clear:

- Borra el campo de búsqueda.
- Restaura la lista completa desde la base de datos.
- Restaura también el estado de orden a su valor inicial (_ordenadoPorFecha = false).

5. Panel de filtros avanzados (búsqueda por proveedor y fechas):

- Se rediseñó completamente para no sobrecargar la parte superior de la pantalla.

- Se implementó un botón `IconButton` con `AnimatedCrossFade` que muestra u oculta los filtros avanzados con animación suave.
- Se mantuvo la funcionalidad completa (búsqueda por proveedor + fechas y búsqueda solo por fechas), accesible bajo demanda.

Problemas detectados y soluciones

- **Advertencia Dart con campos no inicializados (_sinOrdenAplicado):**
 - Solución: se declaró como nullable (`List<HistorialCompra>?`) y se usó con `!` o `??` según el caso.
- **El ordenamiento eliminaba filtros al restaurar la lista:**
 - Se corrigió implementando una copia **solo del estado actual filtrado**, antes de aplicar orden. Así se puede restaurar exactamente la lista que estaba visible.
- **Saturación visual en la cabecera de la pantalla de compras:**
 - Se resolvió con el rediseño animado de filtros avanzados para mostrar u ocultar solo cuando se requiera.

Inicio de planificación del módulo de Eventos

Durante la reunión con Indasor se definieron 4 tipos clave de eventos:

- **ITV** (preaviso 30 días)
- **Mantenimiento** (instalaciones, vehículos) (preaviso 30 días)
- **Revisiones** (alta y baja tensión) (preaviso 30 días)
- **Seguros** (vehículos, instalaciones) (preaviso 60 días)

Para ello se decidió:

- El campo `tipoEvento` en el formulario será reemplazado por un `DropDownButton` con los 4 valores predefinidos, evitando errores de escritura

manual.

- La pantalla de EventosCalendarioScreen será accesible también para técnicos, ya que podrán contribuir activamente a resolver estos eventos: programar citas, buscar aseguradoras, etc.
- Se propuso añadir un nuevo campo estado (pendiente, realizado, cancelado) en la entidad EventoCalendario para marcar aquellos que ya han sido gestionados.
- Los **eventos próximos** se mostrarán en un panel (ej. en HomeAdminScreen) agrupados por cercanía y tipo. Solo se mostrarán eventos pendientes.
- Se diseñará un endpoint `/api/eventos/proximos` que aplicará la lógica de preavisos según el tipo de evento directamente desde el backend.

Plan para mañana

- Modificar la entidad EventoCalendario para añadir el campo estado.
- Crear el nuevo endpoint `/eventos/proximos`.
- Adaptar el formulario de registro con Dropdown de tipo.
- Filtrar eventos por preaviso desde backend.
- Mostrar eventos próximos en HomeAdminScreen y permitir marcar como “realizado” o “cancelado”.

Con estos avances, el módulo de compras queda sólido y profesional, y se sientan las bases para una gestión de eventos mucho más operativa y visualmente clara, ajustada al día a día de Indasor 66.

18/05/2025 — REESTRUCTURACIÓN COMPLETA DEL SISTEMA DE ALBARANES EN INNOVA66

OBJETIVO

Tras reunión con Indasor 66 S.L., se redefine completamente el funcionamiento de los albaranes para ajustarse al uso real de la empresa. El nuevo diseño debe permitir:

- Generación automática del ID del albarán con formato AVI25063
- Creación manual de líneas de Mano de Obra por parte del administrador
- Carga automática de materiales usados desde la actividad técnica
- Edición posterior del albarán para añadir precios y descuentos
- Cálculo automático del total tras valoración
- Visualización de historial de precios de materiales como ayuda para valorar

CAMBIOS PRINCIPALES

MODIFICACIONES EN LA ENTIDAD Albaran.java

- Añadido campo tipoAlbaran (AV, AC, AG) para diferenciar tipos futuros.
- Añadido campo valorado (boolean) para indicar si se ha aplicado precio y descuento.

Añadida relación @OneToMany con LineaAlbaran:

```
java
CopiarEditar
@OneToMany(mappedBy = "albaran", cascade = CascadeType.ALL,
orphanRemoval = true)

private List<LineaAlbaran> lineas;
```

NUEVA ENTIDAD LineaAlbaran.java

Se crea entidad que representa cada línea de un albarán, ya sea Mano de Obra o Material.

- Campos: ref, descripción, unidad, precioUnitario, descuento, totalCalculado
- Relación N:1 con Albaran

NUEVOS DTOs

- LineaAlbaranRequest y LineaAlbaranResponse
- PrecioHistoricoMaterialResponse: representa una compra previa de un material

REESTRUCTURACIÓN DEL AlbaranRequest

- Se elimina campo idAlbaran (ahora es generado automáticamente)
- Se añade lista List<LineaAlbaranRequest> lineas para enviar líneas de MO en la creación

GENERACIÓN AUTOMÁTICA DEL ID DEL ALBARÁN

En AlbaranService, se implementa el método:

java

CopiarEditar

```
private String generarIdAlbaran() {
    String prefijo = "AVI" + (LocalDate.now().getYear() % 100);
    int ultimo = albaranRepository.findUltimoNumeroByPrefijo(prefijo + "%") ?? 0;
    return String.format("%s%03d", prefijo, ultimo + 1);
}
```

Query personalizada en AlbaranRepository:

java

CopiarEditar

```
@Query("SELECT MAX(CAST(SUBSTRING(a.idAlbaran, 6, 3) AS int)) FROM
Albaran a WHERE a.idAlbaran LIKE ?1")
```

```
Integer findUltimoNumeroByPrefijo(String prefijo);
```

CREACIÓN AUTOMÁTICA DE LÍNEAS DE MATERIALES DESDE LA ACTIVIDAD

Dentro de AlbaranService.registrarAlbaran(...), se consulta:

java

CopiarEditar

```
List<ActividadMaterial> materialesUsados =
actividadMaterialRepository.findByActividad(actividad);
```

Cada uno se convierte en una LineaAlbaran con los datos del material, sin precio aún.

EDICIÓN Y ELIMINACIÓN DE LÍNEAS

En LineaAlbaranController se crean endpoints:

- PUT /api/lineas-albaran/{id} para editar línea y recalcular total del albarán
- DELETE /api/lineas-albaran/{id} para eliminar línea y recalcular total

Método auxiliar en LineaAlbaranService:

java

CopiarEditar

```
private void recalcularTotalAlbaran(Albaran albaran) {
    double total = albaran.getLineas().stream()
        .mapToDouble(l -> l.getTotalCalculado() != null ? l.getTotalCalculado() : 0.0)
        .sum();
    albaran.setTotalCalculado(Math.round(total * 100.0) / 100.0);
    albaranRepository.save(albaran);
}
```

HISTORIAL DE PRECIOS DE MATERIALES EN LA EDICIÓN

Se crea el servicio LineaAlbaranMapperService con método mapLineaConHistorial(...) que:

- Verifica si ref corresponde a un material

Consulta el repositorio HistorialCompraRepository con:

```
java
CopiarEditar
List<HistorialCompra> findTop3ByMaterialOrderByFechaDesc(Material material);
```

-
- Devuelve en cada línea del albarán una lista de precios anteriores, con:
 - Fecha
 - Proveedor
 - Precio neto
 - Descuento
 - Precio final

Este historial **no se guarda ni se imprime**, es solo ayuda visual para el administrador en la pantalla de edición.

MODIFICACIÓN DEL AlbaranController

- Inyectado LineaAlbaranMapperService
- En el método GET /api/albaranes/{id} se construye manualmente el AlbaranResponse, usando el mapper para cada línea:

java

CopiarEditar

```
List<LineaAlbaranResponse> lineas = albaran.getLineas().stream()
    .map(lineaAlbaranMapperService::mapLineaConHistorial)
    .toList();
```

RESULTADO FINAL

El backend del módulo de albaranes ha sido completamente reestructurado para reflejar la operativa real de Indasor:

- Se genera el ID automáticamente
- Se cargan automáticamente los materiales usados
- Se permite registrar la mano de obra manualmente
- El administrador valora cada línea posteriormente
- Se recalcula el total al editar/eliminar
- Se ofrece ayuda visual con historial de precios reales

Todo probado desde Postman y preparado para la integración total con Flutter. La lógica está preparada para futuras mejoras como albaranes de compras o gastos.