Facultad
de
Ciencias

DESARROLLO DE MEJORAS UI/UX EN EMI SUITE 4.0
(Development of UI/UX improvements in EMI Suite 4.0)

Trabajo de Fin de Grado
para acceder al

GRADO EN INGENIERÍA INFORMÁTICA

Autor: Pablo Canales Cobo

Directora: Marta Elena Zorrilla Pantaleón

Co-Director: Miguel Sierra Sánchez

Julio – 2025

RESUMEN

El presente trabajo de fin de grado se realizó en la empresa Soincon, dedicada al desarrollo de soluciones tecnológicas para el entorno industrial. El objetivo principal del proyecto fue mejorar la interfaz de usuario (UI – *User interface*) y la experiencia de usuario (UX – *User experience*) de algunos módulos de EMI Suite 4.0 (*Enterprise Manufacturing Intelligence Suite*), una plataforma orientada a la gestión y optimización de procesos industriales en tiempo real.

Para abordar este trabajo, el equipo de Soincon seleccionó una serie de interfaces de la aplicación que requerían una mejora visual y funcional, con el objetivo de modernizar su diseño y facilitar la interacción del usuario. A partir de estos requisitos y después del estudio de buenas prácticas para conformar interfaces visuales, se diseñaron *mockups* interactivos utilizando la herramienta Figma, que sirvieron como referencia para validar las propuestas visuales y estructurales antes de su implementación.

El desarrollo de las mejoras se llevó a cabo utilizando las tecnologías React, JavaScript, Material-UI y CSS, trabajando en un entorno controlado mediante Visual Studio Code. La comunicación con el *back-end* se gestionó a través de la API REST desarrollada por la empresa, lo que permitió mantener la lógica de negocio y la estructura de datos existente. Asimismo, se utilizaron herramientas como GitLab para el control de versiones y Chrome DevTools para la depuración y análisis del comportamiento visual de los componentes.

El proyecto se desarrolló de forma iterativa, incorporando progresivamente los cambios en la interfaz y evaluando su impacto desde el punto de vista de la experiencia de usuario. Todo el desarrollo se llevó a cabo en coordinación con el equipo de *front-end* de la empresa, respetando las directrices y estándares ya establecidos en el sistema.

PALABRAS CLAVE

Interfaz de usuario, experiencia de usuario, usabilidad, diseño de interfaces, Industria 4.0.

ABSTRACT

This final degree project was carried out at Soincon, a company dedicated to developing technological solutions for the industrial environment. The main objective of the project was to improve the user interface (UI) and user experience (UX) of several modules within EMI Suite 4.0, a platform focused on the real-time management and optimization of industrial processes.

To address this project, the Soincon team selected a series of application interfaces that required visual and functional improvements, aiming to modernize their design and enhance user interaction. Based on these requirements and after studying best practices for building visual interfaces, interactive mockups were designed using the Figma tool. These mockups served as a reference for validating both visual and structural proposals prior to implementation.

The improvements were developed using React, JavaScript, Material-UI, and CSS, working within a controlled environment using Visual Studio Code. Communication with the back-end was managed through the company's REST API, which allowed the existing business logic and data structure to be maintained. Additionally, tools such as GitLab were used for version control, and Chrome DevTools for debugging and analyzing the visual behavior of the components.

The project followed an iterative development process, progressively incorporating interface changes and evaluating their impact from a user experience perspective. All development was carried out in coordination with the company's front-end team, adhering to the guidelines and standards already established in the system.

KEYWORDS

User interface, user experience, usability, interface design, Industry 4.0.

ÍNDICE

ÍNDICE DE FIGURAS

LISTADO DE ABREVIATURAS

UI	Interfaz de usuario (del inglés *User Interface*).
UX	Experiencia de usuario (del inglés *User eXperience*).
MES	Sistemas de ejecución de fabricación (del inglés *Manufacturing execution systems).*
MOM	Gestión de operaciones de fabricación (del inglés *Manufacturing Operations Management*).
EMI Suite	Enterprise Manufacturing Intelligence Suite.
OMS	Organización Mundial de la Salud.
DOM	Modelo de Objetos del Documento (del inglés *Document Object Model*).

1. INTRODUCCIÓN

El avance de la digitalización en el ámbito industrial ha generado una transformación significativa en la forma de producir, administrar y tomar decisiones dentro de las plantas de producción. Esta transformación, contextualizada en el concepto de Industria 4.0, ha dado lugar a nuevas exigencias en cuanto al diseño de las herramientas digitales utilizadas en entornos industriales. Ya no es suficiente con que los sistemas sean funcionales: también deben ser intuitivos, accesibles y centrados en el usuario.

Una parte fundamental de esta transformación pasa por los sistemas MES-MOM (*Manufacturing Execution System - Manufacturing Operations Management*), que actúan como núcleo operativo de la planta. Estas plataformas permiten monitorizar y coordinar la producción, el mantenimiento, la calidad y otros procesos clave, en tiempo real. Sin embargo, debido a la complejidad de su funcionalidad y a las condiciones en las que son utilizadas, es frecuente encontrar interfaces sobrecargadas, poco intuitivas o difíciles de usar para el operario medio.

Este trabajo de fin de grado aborda precisamente esa problemática: la necesidad de mejorar la interfaz de usuario (UI) y la experiencia de usuario (UX) de ciertos módulos de la plataforma EMI Suite 4.0, producto de la empresa Soincon. A través de un proceso de análisis, rediseño e implementación, el proyecto se centra en modernizar visualmente la aplicación y optimizar la interacción del usuario, adaptándola a los estándares actuales de diseño y a las condiciones reales del entorno industrial.

1.1 Objetivos

El objetivo general del proyecto es mejorar la interfaz de usuario y la experiencia de usuario en EMI Suite 4.0, mediante la aplicación de principios de diseño centrado en el usuario y tecnologías modernas de desarrollo web.

De este objetivo general se derivan los siguientes objetivos específicos:

1. Investigación sobre principios de UI/UX.

2. Analizar las interfaces existentes en los módulos seleccionados de EMI Suite 4.0, identificando puntos críticos en términos de usabilidad, accesibilidad y estética.

3. Diseñar propuestas de mejora visual y funcional que faciliten la interacción del usuario con el sistema, especialmente en entornos industriales con condiciones adversas.

4. Desarrollar e implementar las nuevas interfaces respetando los estándares de desarrollo del equipo.

5. Documentar el proceso seguido y extraer conclusiones sobre el impacto del rediseño en la usabilidad del sistema.

1.2 Estructura del documento

Este documento se organiza en seis capítulos principales que recogen de forma progresiva los aspectos clave del desarrollo del proyecto:

- *Capítulo 1. Introducción*: Se presenta el contexto del proyecto, sus objetivos y la estructura general del documento.

- *Capítulo 2. Diseño UI / UX*: Se presentan los fundamentos teóricos del diseño de interfaces UI/UX, incluyendo principios generales, aspectos psicológicos y guías. Este capítulo proporciona el marco conceptual que guía el resto del trabajo.

- *Capítulo 3. Metodología y herramientas*: Describe la metodología iterativa adoptada, así como las principales herramientas utilizadas durante las fases de análisis, diseño e implementación.

- *Capítulo 4. Desarrollo*: Recoge con detalle las mejoras realizadas sobre las interfaces seleccionadas, explicando el proceso completo desde el análisis de la versión original hasta el diseño, implementación y validación de los cambios.

- *Capítulo 5. Validación y resultados*: Se analiza el impacto de las mejoras mediante pruebas funcionales y de usabilidad. Se presentan los resultados obtenidos tras aplicar las nuevas interfaces y se evalúa su efectividad en base a criterios definidos previamente.

- *Capítulo 6. Conclusiones y líneas de trabajo futuro*: Resume las conclusiones alcanzadas, las lecciones aprendidas y propone posibles líneas de trabajo futuras que puedan continuar o ampliar las mejoras implementadas.

Al final del documento se incluye la bibliografía utilizada como soporte teórico y técnico, así como una serie de anexos con información adicional relevante para complementar el contenido principal.

2. DISEÑO UI / UX

En el contexto del desarrollo de aplicaciones, el diseño de la interfaz de usuario y la experiencia de usuario son dos disciplinas fundamentales que trabajan conjuntamente para garantizar productos digitales funcionales, atractivos y fáciles de utilizar.

La interfaz de usuario se refiere a todos los elementos visuales e interactivos con los que el usuario establece contacto, como botones, menús, iconos, tipografías y colores. Su objetivo principal es facilitar la interacción del usuario con el sistema de forma intuitiva y eficiente.

Por otro lado, la experiencia de usuario abarca la percepción global que un usuario tiene al interactuar con un sistema o producto. Incluye factores como la facilidad de aprendizaje, la eficiencia de uso, la satisfacción emocional y la accesibilidad. Un buen diseño UX no solo busca que un sistema sea usable, sino también que sea agradable y significativo para el usuario.

Aunque UI y UX son conceptos distintos, su relación es inseparable. Un diseño de UI pobre puede deteriorar una buena experiencia de usuario, mientras que una excelente UI, sin considerar la experiencia completa del usuario, también puede resultar en un producto fallido. Tal como afirma Garrett (2010), “el diseño de la experiencia de usuario incluye todos los aspectos de la interacción del usuario con la empresa, sus servicios y sus productos”.

En el ámbito industrial, el diseño UI/UX cobra una especial importancia. Los usuarios suelen trabajar en entornos de alta presión donde la eficiencia, la claridad visual y la facilidad de uso son determinantes para evitar errores y maximizar la productividad. Un diseño cuidado puede reducir significativamente el tiempo de formación necesario para nuevos operarios, minimizar los fallos humanos y mejorar la aceptación del sistema por parte del personal de planta.

Así, comprender los principios que rigen el diseño UI/UX resulta esencial para abordar con éxito cualquier proceso de mejora de interfaces industriales, garantizando que las soluciones implementadas no solo sean técnicamente correctas, sino también centradas en las necesidades reales de los usuarios.

Aunque no existe una lista universal de principios de diseño UI/UX, existen varios ampliamente aceptados por la comunidad de diseño, respaldados por guías oficiales como *Material Design* (Google), *Human Interface Guidelines* (Apple) o la heurística de usabilidad de Jakob Nielsen.

2.1 Diseño UI

El diseño de la interfaz de usuario se centra en construir interfaces visuales intuitivas, funcionales y estéticamente coherentes, teniendo como objetivo facilitar la interacción entre el usuario y el sistema, asegurando que cada elemento visual transmita claramente su propósito y comportamiento esperado. En las aplicaciones de gestión de planta desplegadas en contextos industriales, estos principios adquieren una relevancia importante, dado que los operarios deben interactuar con sistemas bajo condiciones exigentes, como iluminación deficiente, ruido elevado o la necesidad de usar guantes de protección.

2.1.1 Principios generales de diseño UI

A continuación, se describen los principios esenciales que deben guiar el diseño de una interfaz de usuario eficaz:

- Simplicidad -

El principio de simplicidad en el diseño de interfaces significa mantener una interfaz limpia, ordenada y sencilla, buscando reducir al mínimo la complejidad innecesaria, presentando únicamente la información y funcionalidades esenciales para que el usuario pueda interactuar con el sistema de forma clara, rápida y eficiente. Una interfaz simple no es sinónimo de limitada o básica, sino de bien pensada: cada elemento que aparece en pantalla debe tener una función específica y aportar valor a la tarea que se está realizando.

Diseñar con simplicidad implica tomar decisiones conscientes sobre qué mostrar, cuándo y cómo. Cuando una interfaz es simple, los usuarios no necesitan leer instrucciones extensas ni realizar múltiples pasos para alcanzar su objetivo: todo está dispuesto de forma natural y accesible.

- Consistencia -

La consistencia es uno de los principios fundamentales del diseño de interfaces de usuario, ya que permite al usuario construir modelos mentales estables sobre cómo funciona una aplicación o sistema. Una interfaz consistente presenta patrones de diseño repetibles, comportamientos predecibles y una organización coherente de sus elementos, lo que facilita el aprendizaje, reduce la carga cognitiva y minimiza la aparición de errores.

A nivel visual, la consistencia implica el uso coherente de colores, tipografías, iconos, estilos de botones y espaciados. Por ejemplo, si un botón de confirmación es de color verde y tiene forma rectangular con esquinas redondeadas, mantener ese estilo en todas las pantallas evita confusiones y mejora la identificación rápida de acciones clave. A nivel funcional, la consistencia asegura que una misma acción tenga siempre el mismo resultado y que elementos con funciones similares estén agrupados

y etiquetados de forma coherente, consiguiendo que el usuario deje de pensar en la interfaz y se centre exclusivamente en sus tareas.

En la Figura 1 se puede observar una comparación entre un diseño consistente y otro inconsistente. Este ejemplo muestra cómo los cambios en la disposición visual, los elementos de navegación o los estilos pueden generar confusión en el usuario y afectar a la experiencia de uso.

Figura 1: Diferencia entre un diseño consistente y un diseño no consistente.
Fuente: https://maze.co/

- Retroalimentación inmediata -

El principio de retroalimentación en el diseño de interfaces de usuario se refiere a la capacidad del sistema para informar al usuario sobre el resultado de sus acciones. Cada vez que un usuario interactúa con una interfaz, ya sea haciendo clic en un botón, introduciendo datos o cargando una página, el sistema debe proporcionar una respuesta perceptible que confirme que la acción ha sido reconocida y procesada.

Esta retroalimentación puede manifestarse de diferentes formas, como cambios visuales (por ejemplo, un botón que se ilumina al hacer clic), mensajes de confirmación, animaciones, sonidos o vibraciones.

Una retroalimentación adecuada contribuye significativamente a la usabilidad, ya que reduce la incertidumbre y aumenta la sensación de control. Sin ella, los usuarios pueden sentirse confundidos o dudar de si sus acciones han tenido efecto, lo que puede derivar en errores o frustración.

- Jerarquía visual -

La jerarquía visual en el diseño de interfaces se basa en guiar la atención del usuario hacia los elementos más importantes de la pantalla mediante el uso intencionado de atributos visuales como el tamaño, el color, el contraste, la ubicación o la tipografía. El objetivo es establecer un orden de lectura natural que facilite la comprensión y la interacción eficiente con la interfaz.

Este principio se apoya en la psicología de la percepción visual y en leyes como la de escaneo en forma de F o la de atención selectiva, según las cuales el usuario explora la interfaz de forma no lineal, deteniéndose primero en los elementos más destacados. Por ello, es esencial que los componentes clave como botones de acción, indicadores de estado o mensajes críticos, ocupen posiciones prominentes y estén diseñados para sobresalir del resto de la interfaz.

2.1.2 Elementos visuales del diseño de interfaces

Además de seguir principios generales como los descritos anteriormente, un buen diseño de interfaz debe cuidar los aspectos visuales que condicionan directamente la legibilidad, comprensión y experiencia del usuario. A continuación, se describen los principales componentes visuales a tener en cuenta:

- Uso del color -

El color es una herramienta poderosa en la interfaz, ya que tiene un gran impacto en cómo los usuarios perciben y experimentan la interfaz. Utilizado estratégicamente, el color dirige la atención hacia elementos clave, como botones de acción o mensajes importantes, y transmite emociones: el azul, por ejemplo, despierta tranquilidad y confianza, mientras que el rojo puede señalar urgencia o advertencia. En un entorno industrial el verde indica funcionamiento normal o condiciones seguras; el rojo, se asocia a alertas o paradas de emergencia, mientras que el amarillo es usado para advertencias.

- Tipografía -

Elegir correctamente la familia tipográfica, su tamaño, el grosor y el espaciado puede marcar la diferencia entre una interfaz clara y comprensible y una que genere confusión o fatiga visual.

Las fuentes sans-serif, como Roboto, Open Sans o Helvetica, suelen ser preferidas en entornos digitales por su claridad en pantallas. El tamaño de la fuente debe adaptarse al dispositivo y al entorno de uso, procurando que el texto pueda leerse sin esfuerzo desde una distancia razonable o en pantallas táctiles industriales. La jerarquía tipográfica, es decir, la diferenciación visual entre títulos, subtítulos y cuerpo de texto debe ser clara y constante, permitiendo que el usuario entienda rápidamente qué partes del contenido son más relevantes.

- Agrupación visual y alineación -

La agrupación visual y la alineación son principios esenciales para estructurar correctamente una interfaz y facilitar la interpretación rápida de la información. Uno de los fundamentos teóricos que respalda su uso es la Ley de Proximidad de la psicología de la Gestalt, según la cual los elementos situados cerca unos de otros tienden a percibirse como parte de un mismo grupo. Aplicar este principio permite organizar visualmente los componentes de una interfaz en bloques coherentes, ayudando al usuario a entender de un vistazo qué elementos están relacionados funcionalmente.

La alineación, por su parte, aporta una estructura visual sólida y coherente. Cuando los elementos están alineados entre sí, se genera un patrón visual predecible que guía la mirada y refuerza la jerarquía de contenidos. Una interfaz desalineada, en cambio, transmite desorden e imprecisión, dificultando la lectura fluida de los elementos y generando fatiga visual.

En la Figura 2 se muestran dos ejemplos: el primero, correctamente alineado y estructurado, permite una lectura intuitiva y clara; mientras que el segundo, sin alineación, resulta caótico y dificulta la percepción rápida de los elementos relacionados.

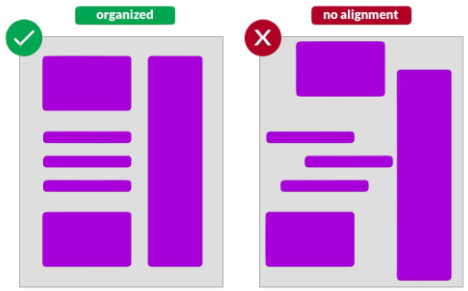

Figura 2: Comparativa de una interfaz agrupada y alineada.

- Longitud de párrafos y densidad de información -

El contenido textual debe ser tratado con especial cuidado en el diseño de interfaces, ya que una redacción inadecuada puede dificultar la interpretación y generar ambigüedades. En contextos industriales, donde los operarios deben comprender instrucciones, mensajes de error o indicadores de estado de forma rápida, es recomendable que los textos sean breves, concretos y orientados a la acción. Los párrafos extensos, con estructuras complejas o terminología ambigua, pueden ralentizar la toma de decisiones o inducir a fallos operativos. Es preferible utilizar frases cortas, en voz activa, y con verbos que indiquen claramente la acción esperada.

- Espaciado y uso del blanco -

Es un recurso fundamental en el diseño de interfaces que, lejos de ser un vacío visual, actúa como una herramienta organizativa que mejora la comprensión, la

legibilidad y la estética del contenido. Un uso adecuado del espaciado permite al usuario identificar con claridad cada sección, elemento o grupo funcional, disminuyendo la carga cognitiva y facilitando la navegación. El espacio en blanco actúa como un separador natural entre bloques de información, previniendo el colapso visual que puede producirse en interfaces recargadas o desorganizadas.

2.2 Diseño UX

Según la ISO 9241-11:2018 (International Organization for Standardization, 2018), la experiencia de usuario se refiere a las "percepciones y respuestas de la persona resultantes del uso y/o el uso anticipado de un producto, sistema o servicio". Además, incluye "todas las emociones, creencias, preferencias, percepciones, respuestas físicas y psicológicas, comportamientos y logros de los usuarios que ocurren antes, durante y después del uso".

En este contexto, el concepto de usabilidad, definido en la ISO 9241-210:2019 (International Organization for Standardization, 2019), se integra como un componente esencial de la experiencia de usuario, refiriéndose al "grado en que un producto puede ser utilizado por usuarios específicos para lograr objetivos concretos con eficacia, eficiencia y satisfacción, en un determinado contexto de uso".

Mientras la usabilidad se centra en aspectos operativos clave como la eficacia, la eficiencia y la satisfacción, la experiencia de usuario llega a incluir también las dimensiones emocionales, subjetivas y contextuales de la interacción.

Por tanto, el diseño UX no se limita a optimizar la usabilidad del sistema, sino que busca garantizar que la experiencia completa, desde el primer contacto hasta el uso prolongado, no solo sea efectiva y eficiente, sino también significativa, placentera y adaptada a las expectativas y necesidades reales de los usuarios.

En la Figura 3 se representa cómo la experiencia del usuario se construye dinámicamente a partir de sus experiencias previas, su motivación y el contexto en el que interactúa, generando nuevas expectativas en el tiempo (T en la Figura 3). A su vez, la Figura 4 muestra que dicha experiencia depende de tres factores principales: el estado interno del usuario, el entorno o contexto, y las características del sistema.

Figura 3: Modelo dinámico de la experiencia del usuario a lo largo del tiempo (Mäkelä, A. & Fulton Suri, J. 2001).

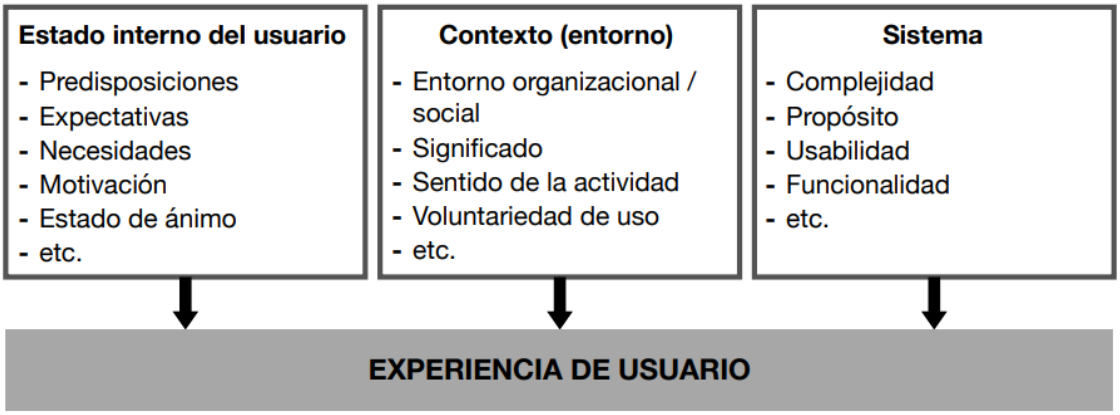

Figura 4: Factores que influyen en la experiencia de usuario (Hassenzahl, M. & Tractinsky, N. 2006).

2.2.1 Principios generales de diseño UX

- Enfoque en el usuario -

El diseño UX siempre debe partir de una premisa clara: poner a las personas en el centro. Y no solo pensando en lo que necesitan, sino en cómo se sienten al interactuar con el producto. Para lograrlo, el diseño centrado en el usuario, según la norma ISO 9241-210:2010 (International Organization for Standardization, 2010), propone involucrarlos desde el principio, escucharlos, observar cómo trabajan y validar cada paso con ellos, no con suposiciones.

En entornos industriales, este enfoque es aún más importante. Aquí, detalles como el uso de guantes, el ruido, la iluminación o la presión del tiempo influyen directamente en la experiencia. Entender ese contexto es lo que convierte un diseño correcto en uno realmente útil, cercano y humano.

- Accesibilidad -

Diseñar interfaces accesibles garantiza que todas las personas, independientemente de sus capacidades físicas, sensoriales o cognitivas, puedan interactuar con el sistema de manera eficiente, segura y satisfactoria. Según datos de la OMS (Organización Mundial de la Salud, 2023, 2025), hay alrededor de 285 millones de personas en el mundo con discapacidad visual, entre 110 y 190 millones de adultos presentan dificultades significativas de movilidad y 430 millones padecen pérdida auditiva discapacitante.

Pero más allá de estas cifras, diseñar para todos no solo beneficia a quienes tienen alguna discapacidad, sino que mejora la experiencia de uso en general. Un botón con suficiente contraste, por ejemplo, es más fácil de ver para cualquier persona en un entorno con mala iluminación. Un texto claro y sencillo no solo ayuda a quienes tienen dificultades cognitivas, también reduce la carga mental de todos los usuarios.

- Usabilidad -

La usabilidad es uno de los principios más importantes del diseño UX. Un diseño usable debe ser intuitivo, fácil de navegar y minimizar la carga cognitiva del usuario, permitiéndole enfocarse en sus objetivos sin distracciones ni complicaciones.

Cuando un producto cumple este principio de usabilidad, los usuarios pueden alcanzar sus metas de manera rápida, precisa y con una experiencia satisfactoria. Esto evita que se sientan torpes, inseguros o frustrados al interactuar con la interfaz, lo que mejora su percepción del producto y fomenta su uso.

- Eficiencia -

La eficiencia, en diseño UX, no va solo de velocidad, sino de eliminar fricciones innecesarias que entorpecen la experiencia. Se trata de que el usuario pueda hacer lo que necesita con el mínimo esfuerzo, sin pasos redundantes ni dudas que lo hagan perder tiempo.

Esto implica optimizar tanto la navegación como los procesos, mostrar la información adecuada en el momento oportuno y anticiparse a las necesidades del usuario. En lugar de sobrecargar de opciones o pasos intermedios, una interfaz eficiente guía con naturalidad hacia el objetivo del usuario.

2.2.2 Principios psicológicos y heurísticos en el diseño UX

Y es que estos principios no aparecen de la nada. No son simples ocurrencias ni reglas impuestas al azar. Detrás de cada uno de ellos hay una base sólida, construida a partir de cómo las personas realmente ven, piensan, sienten e interactúan con lo que tienen delante.

En lo que a la experiencia de usuario se refiere, la psicología, especialmente la de la percepción, la cognición y la interacción persona-computador, es imprescindible. Gracias a ella, conceptos que podrían parecer abstractos se transforman en guías claras y muy útiles para tomar decisiones de diseño más acertadas. Son esas pequeñas claves que explican por qué agrupar elementos similares facilita la comprensión, o por qué ofrecer retroalimentación inmediata puede reducir la frustración del usuario.

- Teoría de la Gestalt -

Es una de las principales teorías desarrolladas para explicar los mecanismos visuales y perceptivos del cerebro humano. Esta teoría, desarrollada por psicólogos alemanes a principios del siglo XX, permite comprender mejor cómo las personas organizan visualmente la información para darle sentido. Sus principios básicos examinan cómo el cerebro humano combina elementos individuales para formar una estructura completa.

A continuación, se detallan algunas de las leyes de la Gestalt más relevantes en el diseño UX (Aliyev, 2024; Fernández Casado, 2021):

- **Principio de Figura-Fondo:** Este principio se basa en nuestra capacidad de distinguir un elemento principal (la figura) del entorno que lo rodea (el fondo). En diseño UX, este principio es clave para guiar la atención del usuario hacia lo importante, como botones, textos o alertas, creando contraste visual claro y jerarquías comprensibles. Cuando figura y fondo no están bien definidos, puede generarse confusión o incluso una mala interpretación de la interfaz.

  La Figura 5 ilustra el principio de figura-fondo mediante la Copa de Rubin, un ejemplo clásico de cómo un mismo diseño puede percibirse de dos maneras distintas: como un objeto central o como los perfiles de dos rostros.

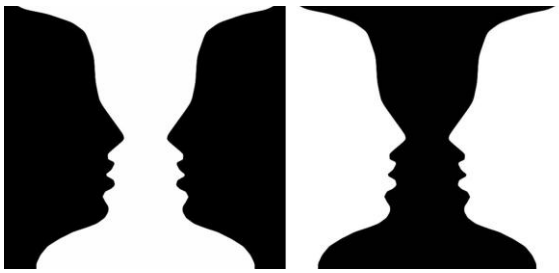

Figura 5: Ejemplo principio de figura-fondo. Copa de Rubin.

- **Principio de Proximidad:** Los elementos más próximos entre sí tienden a verse como una única entidad global, incluso si las formas y tamaños son muy diferentes. En diseño UX, esto se traduce en organizar la información de forma que la agrupación visual tenga sentido para el usuario. Por ejemplo, si los campos de un formulario están muy separados, pueden parecer independientes o desconectados, dificultando la experiencia.

  En el ejemplo que se muestra en la Figura 6, comprobamos que la única diferencia entre el conjunto de la izquierda y el de la derecha es la proximidad entre los elementos. Sin embargo, nuestra percepción visual interpreta la imagen de la derecha como tres grupos distintos, mientras que la de la izquierda se percibe como una única unidad.

Figura 6: Ejemplo principio de proximidad.

- **Principio de Semejanza:** Elementos similares tienden a ser percibidos como parte de una misma forma o elemento que lo engloba todo. Esto se produce porque nuestro cerebro tiende a agrupar los elementos que poseen alguna propiedad visual común como el color, el tamaño, la forma, el brillo o el

movimiento. Este principio se utiliza ampliamente en el diseño UX/UI para garantizar la coherencia.

En la Figura 7 los círculos son equidistantes entre sí y tienen el mismo tamaño, pero nuestra mente los agrupa, en este caso, por color, distinguiendo dos conjuntos: el azul y el negro.

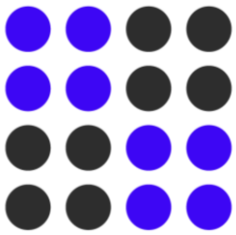

Figura 7: Ejemplo principio de semejanza.

- **Principio de Continuidad:** Los elementos que forman patrones o que están colocados de una manera determinada según un sentido o dirección tienden a ser agrupados. El ojo humano va a seguir siempre el camino visual más suave, menos forzado y más coherente. En diseño UX, esto se aplica al organizar componentes de forma coherente y alineada: menús, formularios, procesos paso a paso o flujos de navegación que guían al usuario con naturalidad.

  La Figura 8 ilustra este principio mediante dos series de puntos que se cruzan en un punto central. Aunque los puntos azules y negros están mezclados en esa intersección, nuestro ojo va a querer seguir la línea recta o curva, de un extremo a otro, aunque la línea cambie de color.

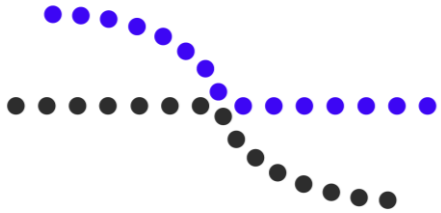

Figura 8: Ejemplo principio de continuidad.

- **Principio de Cierre:** El cerebro tiende a completar las figuras abiertas y próximas entre sí, ya que las percibe como un conjunto, aunque estén incompletas.

  Un ejemplo claro se observa en la iconografía utilizada en bibliotecas de diseño como Material UI, Figura 9. A pesar de que algunos iconos no están completamente cerrados o definidos con líneas continuas, el usuario los interpreta fácilmente gracias al principio de cierre, percibiendo figuras completas.

Figura 9: Ejemplo principio de cierre. Iconografía Material UI.

- Principios de Shneiderman -

En 1998, el investigador Ben Shneiderman propuso un conjunto de reglas conocidas como las "Ocho reglas de oro" (Shneiderman, 1998). Aunque fueron formuladas hace más de dos décadas, estos principios siguen siendo una guía clara y vigente para crear interfaces intuitivas, eficientes y centradas en el usuario.

Cada uno de estos principios apunta a reducir la carga cognitiva, prevenir errores y facilitar la interacción con los sistemas, por lo que encajan perfectamente con los pilares del diseño UX que hemos definido previamente.

- **Esforzarse por conseguir consistencia:** Deben usarse secuencias consistentes de acciones en situaciones similares. Este principio habla sobre la importancia de utilizar la misma terminología en las indicaciones, los menús y las pantallas de ayuda. “Las interfaces consistentes permiten que el usuario se familiarice con el sistema, lo que le ayuda a utilizarlo correctamente” (Cha A.P., Romli A., 2010).

- **Crear atajos para los usuarios frecuentes:** Este requerimiento se vuelve una necesidad a medida que aumenta la frecuencia de uso. Los usuarios, según van aumentando su conocimiento sobre la interfaz o sistema, van deseando disminuir el número de interacciones y el ritmo de la interacción. Por esta razón, se deben utilizar abreviaturas, teclas de función, comandos ocultos y macro instalaciones. Un ejemplo clásico y ampliamente utilizado son los atajos de teclado como Ctrl + C para copiar y Ctrl + V para pegar.

- **Ofrecer feedbacks:** Para cada acción realizada por el usuario, el sistema debe mostrarle la retroalimentación para indicarle en qué estado se encuentra. Si las acciones son frecuentes y menores, la respuesta puede ser poco detallada. Si las acciones son poco frecuentes e importantes, la respuesta debe ser más sustancial. Además, los tiempos de respuesta deben ser adecuados y que no hagan que se pierda la atención del usuario.

- **Diseñar el diálogo para mostrar trabajo pendiente:** Todas las secuencias de acciones deben organizarse en grupos con un principio, un medio y un final y se les debe informar a los usuarios del estado de dichas acciones. Shneiderman (1998) asegura que dar una retroalimentación informativa a los usuarios, después de terminar una acción en una página, les brinda la satisfacción de haber logrado algo, una sensación de alivio o una señal para dejar de pensar en planes alternativos en caso de que se presente un inconveniente.

- **Ofrecer una gestión sencilla de los errores:** En la medida de lo posible, se debe diseñar el sistema para que el usuario no pueda cometer ningún error grave. Si se produce un error, el sistema debe ser capaz de detectarlo y ofrecer unos mecanismos simples y comprensibles para manejar dicho error.

- **Permitir una fácil recuperación de acciones:** Se debe permitir la reversión de las acciones, de entradas de datos y de procesos de forma sencilla. "Esta función alivia la ansiedad, ya que el usuario sabe que los errores se pueden revertir y, por lo tanto, fomenta la exploración de opciones desconocidas" (Cha A.P., Romli A., 2010).

- **Proporcionar control al usuario:** Un buen diseño UX no impone, sino que acompaña. Y es que cuando los usuarios sienten que tienen el control sobre la interfaz, y no al revés, su experiencia mejora notablemente. Este principio defiende la idea de que las personas deben poder iniciar, interrumpir, rehacer o deshacer acciones con facilidad. Por ejemplo, permitir cancelar un proceso de carga, retroceder a una pantalla anterior o deshacer un cambio accidental genera confianza y reduce la frustración.

- **Reducir la necesidad de memorización por parte del usuario:** De acuerdo con la neurociencia y la psicología, las personas tienen una capacidad limitada para el procesamiento de información en la memoria a corto plazo, por lo tanto, es clave evitar las interfaces en las que los usuarios deben recordar la información de una pantalla para luego utilizarla en otra.

- Heurística de Nielsen -

Además de los principios psicológicos y de interacción, el diseño UX también se apoya en guías prácticas que ayudan a detectar y prevenir errores comunes. Estas heurísticas no son normas rígidas, sino principios flexibles que orientan el diseño hacia la claridad, la eficiencia y la facilidad de uso.

Jakob Nielsen, autoridad reconocida en el campo de la usabilidad y cofundador de la consultora Nielsen Norman Group propone un conjunto de principios generales que sirven como guía para evaluar la calidad de uso de un sistema, especialmente en etapas tempranas del diseño (Nielsen, 1994):

- **Visibilidad del estado del sistema:** El sistema debe mantener siempre informados a los usuarios sobre lo que está sucediendo, mediante retroalimentación apropiada dentro de un período de tiempo razonable.

- **Coincidencia entre el sistema y el mundo real:** El sistema debe utilizar un lenguaje al que los usuarios estén habituados, con una jerga o terminología que les sea conocida y legible, en lugar de términos o mensajes genéricos que se

utilizan en el sistema, para que al usuario se sienta cómodo al utilizar el sistema.

- **Control y libertad al usuario:** Los usuarios pueden cometer errores y por eso ninguna acción que el usuario pueda llevar a cabo en una interfaz debería ser irreversible, sobre todo aquellas que pueden tener consecuencias negativas para el usuario mismo.

- **Consistencia y estándares:** El sistema tiene que seguir unas normas coherentes y las convenciones de la plataforma para que el usuario pueda prever cómo interactuar sin necesidad de preguntarse cómo hacerlo o aprender nuevas acciones.

- **Prevención de errores:** Los buenos mensajes de error son importantes, pero los mejores diseños previenen cuidadosamente la aparición de problemas desde el principio.

- **Reconocer en lugar de recordar:** El sistema debe minimizar la información que el usuario debe recordar mostrándosela a través de objetos, acciones u opciones. La información necesaria para usar el sistema debe ser visible o fácilmente recuperable cuando se necesite.

- **Flexibilidad y eficiencia de uso:** Se deben proporcionar métodos abreviados o atajos de teclado para que tanto los usuarios novatos como los expertos aprovechen la capacidad del sistema y se puedan adaptar sin importar el nivel de conocimiento de la plataforma.

- **Estética y diseño minimalista:** Las interfaces no deben contener información irrelevante o rara vez necesaria. Cada unidad de información adicional en una interfaz compite con las unidades de información relevantes y reduce su visibilidad relativa.

- **Ayudar al usuario a reconocer, diagnosticar y recuperarse de los errores:** Los mensajes de error deben expresarse en un lenguaje sencillo (sin códigos de error), indicar con precisión el problema y sugerir una solución de forma constructiva.

- **Ayuda y documentación:** Aunque es mejor que el sitio web o aplicación pueda ser usado sin ayuda, puede ser necesario proveer cierto tipo de ayuda. Si este requerimiento se vuelve necesario, la ayuda debe ser fácil de localizar, se deben especificar los pasos necesarios y no ser muy extensa.

3. METODOLOGÍA Y HERRAMIENTAS

Este capítulo describe el enfoque metodológico seguido durante el desarrollo del proyecto, basado en un proceso iterativo orientado a la mejora continua de la interfaz. Asimismo, se detallan las herramientas utilizadas tanto en el diseño como en la implementación.

3.1 Metodología adoptada

Este TFG se ha desarrollado siguiendo una metodología de trabajo iterativa como se muestra en la Figura 10, la cual ha permitido avanzar de forma progresiva, evaluando y refinando las soluciones a lo largo del tiempo. Este enfoque resulta especialmente útil en proyectos donde el diseño y la experiencia del usuario desempeñan un papel central, ya que facilita una evolución constante basada en el análisis y la mejora continua.

Aunque no fue posible contar con la participación directa de usuarios finales, se mantuvo siempre presente una mentalidad centrada en el usuario, tomando como referencia sus posibles necesidades, limitaciones y el contexto real de uso.

La metodología se desarrolló a través de las siguientes etapas:

1. Análisis de la interfaz: revisión de la aplicación existente, identificando fricciones, carencias de usabilidad y oportunidades de mejora visual.
2. Definición de requisitos: extracción de los objetivos a partir del análisis, con base en los principios de diseño UI/UX definidos en el marco general.
3. Diseño de soluciones: elaboración de propuestas de mejora y diseño de *mockups* en Figma como base visual para la implementación.
4. Implementación: desarrollo e integración progresiva de las mejoras propuestas en la interfaz, integrándolas progresivamente en el entorno real de la aplicación.
5. Evaluación: verificación del funcionamiento y la coherencia de las soluciones desarrolladas mediante pruebas y revisiones.

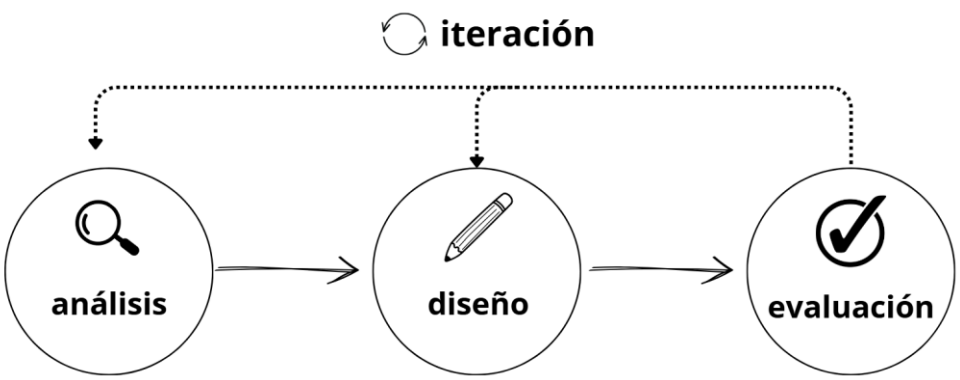

Figura 10: Metodología iterativa.

3.2 Herramientas utilizadas

3.2.1 Herramientas de diseño

Figma: Es una herramienta de diseño de interfaces basada en la nube que permite crear prototipos interactivos y colaborar en tiempo real con otros miembros del equipo. Se utilizó para diseñar *mockups* de las nuevas interfaces, facilitando la validación de las propuestas visuales antes de su implementación.

3.2.2 Entorno de desarrollo

Visual Studio Code (VSCode): Editor de código ligero y multiplataforma, ampliamente utilizado en entornos de desarrollo web. Su sistema de extensiones y personalización lo convierte en una opción flexible para trabajar con tecnologías como React y JavaScript.

3.2.3 Entorno de ejecución y gestión de dependencias

Node.js: Entorno de ejecución de JavaScript en el servidor, necesario para ejecutar y compilar las aplicaciones basadas en React. Permite utilizar herramientas modernas de desarrollo *front-end*.

npm (Node Package Manager): Gestor de paquetes utilizado para instalar y mantener las dependencias del proyecto, como librerías, herramientas de desarrollo y componentes de terceros. Su uso asegura la trazabilidad y consistencia del entorno de trabajo.

3.2.4 Tecnologías *front-end*

React: Biblioteca de JavaScript utilizada para construir interfaces de usuario basadas en componentes reutilizables. Facilita el desarrollo de aplicaciones interactivas con una estructura modular y mantenible.

JavaScript: Lenguaje de programación utilizado para la lógica del lado del cliente. Permite el manejo dinámico del DOM (*Document Object Model*), la interacción con la API y la gestión de estados dentro de la aplicación.

CSS: Lenguaje de estilos utilizado para definir la apariencia visual de los elementos. Se empleó en combinación con Material-UI para personalizar y adaptar el diseño de los componentes.

Material-UI: Librería de componentes basada en las directrices de Material Design de Google. Proporciona un conjunto de elementos visuales listos para usar, que aseguran coherencia estética y buena experiencia de usuario.

Highcharts: Librería de JavaScript orientada a la visualización de datos en la web mediante gráficos interactivos. Tiene una API altamente personalizable, con soporte para interacciones como *tooltips*, *zoom*, exportación o actualización en tiempo real. Además, se adapta a múltiples dispositivos y navegadores, lo que la hace ideal para aplicaciones con necesidades de representación visual de datos complejos.

3.2.5 Control de versiones y análisis de rendimiento de interfaz

GitLab: Plataforma de control de versiones basada en Git, que permite gestionar el código fuente, trabajar con ramas y documentar el proceso de desarrollo en colaboración con otros miembros del equipo.

Chrome DevTools: Conjunto de herramientas integradas en el navegador Google Chrome, utilizadas para depurar, analizar y optimizar el comportamiento de la aplicación durante su ejecución.

Cypress: Herramienta de testing automatizado utilizada para realizar pruebas *end-to-end* sobre la aplicación. Permite simular interacciones reales del usuario en el navegador, validar comportamientos esperados y detectar errores funcionales de forma automática.

3.2.6 Comunicación con el *back-end*

API REST de la empresa: Interfaz de comunicación desarrollada por Soincon, utilizada para obtener y enviar datos entre el *front-end* y el sistema central de EMI Suite 4.0. Permite mantener la lógica de negocio y trabajar con datos reales.

Swagger.io: Conjunto de herramientas de código abierto para el diseño, documentación y prueba de APIs RESTful. Swagger incluye interfaces interactivas que permiten explorar los distintos *endpoints*, realizar peticiones simuladas y validar esquemas de datos. Es especialmente útil para mantener la coherencia en proyectos colaborativos y garantizar una comunicación clara entre *front-end* y *back-end*.

4. DESARROLLO

En este capítulo se detallan las distintas fases del proceso de rediseño e implementación de las interfaces seleccionadas. A partir del análisis de las versiones originales, se identificaron sus principales limitaciones y se definieron los requisitos necesarios para su mejora. Posteriormente, se presentan los diseños propuestos y las decisiones técnicas adoptadas durante su implementación, destacando los criterios de usabilidad y experiencia de usuario aplicados en cada caso.

4.1 Interfaz Kanban de seguimiento de órdenes

La primera interfaz corresponde al tablero Kanban del módulo de Visual Tracking dentro de EMI Suite 4.0 (Figura 11). Esta vista tiene como objetivo representar visualmente el estado de las órdenes de fabricación a lo largo de las distintas fases del proceso productivo de una empresa industrial.

El sistema divide las órdenes en columnas que representan etapas desde el pedido y confirmación, hasta la planificación, producción, calidad y distintas etapas de validación, permitiendo a los usuarios ver de forma simultánea en qué punto del flujo se encuentra cada orden. Cada tarjeta representa una orden de fabricación, e incluye información clave como el código, nombre del cliente, la descripción y las etiquetas.

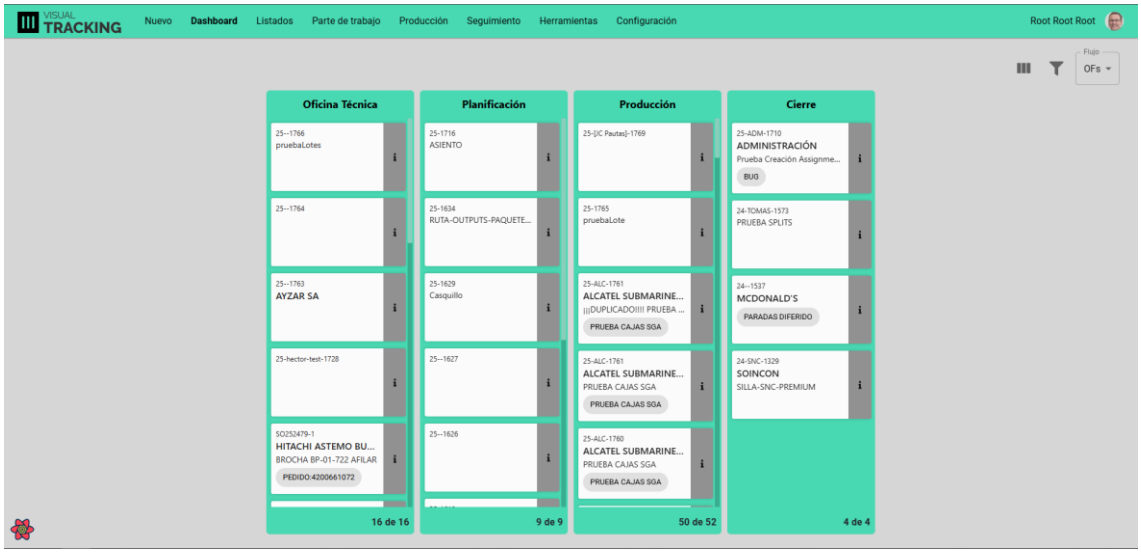

Figura 11: Interfaz Kanban original.

4.1.1 Análisis de la interfaz original

A primera vista, la interfaz original presenta un diseño simple y funcional que cumple con su objetivo principal: mostrar el estado de distintos trabajos distribuidos a lo largo de varias fases mediante un sistema de columnas tipo Kanban. La estructura general permite entender rápidamente que se trata de un flujo de producción, y el uso de tarjetas facilita la división visual de tareas. Sin embargo, al analizarla con más

detalle se pueden identificar varias limitaciones que dificultan la claridad y el uso ágil de la interfaz.

Uno de los principales problemas es la falta de diferenciación entre las tarjetas que representan las órdenes de fabricación. Todas comparten el mismo aspecto visual, sin variaciones de color, forma o jerarquía que permitieran al usuario distinguir entre distintos tipos de órdenes o prioridades.

Adicionalmente, el uso de un fondo verde tan llamativo compite visualmente con las tarjetas de órdenes de fabricación llegando a vulnerar el principio de figura-fondo de la psicología de la Gestalt, ya que el fondo capta la atención en lugar de permanecer discreto. Como resultado, el usuario tiene dificultades para centrar su atención en los elementos realmente importantes del tablero Kanban, lo que afecta tanto a la comprensión como a la fluidez visual.

A esto se suma la visibilidad reducida de los botones situados en la parte superior derecha. Estos botones, de tamaño pequeño con un color similar al fondo, resultan difíciles de detectar. Además, su ubicación junto al selector de flujo (cuyo ancho variaba según el nombre del flujo seleccionado) genera una distribución inestable y poco consistente.

Otro problema detectado es el reducido tamaño de las tarjetas. Su diseño compacto limita la cantidad de información visible, lo que provoca que muchos textos aparezcan truncados. Esto dificulta la lectura y aumenta la carga cognitiva del usuario.

La disposición general del tablero también presenta dificultades. La separación entre columnas es demasiado estrecha, lo que genera una sensación de apelmazamiento y dificulta percibir con claridad la estructura del flujo de trabajo. Esta falta de espacio compromete la lectura visual en bloque y reduce la capacidad del usuario para identificar rápidamente en qué fase se encontraba cada grupo de tareas.

Finalmente, los contadores que indican el número de órdenes de fabricación que hay en cada columna están ubicados en la parte inferior de cada fase, desconectados visualmente del encabezado. Esta separación incumple el principio de proximidad, ya que divide información que conceptualmente debe estar unida: el nombre de la fase y la cantidad de elementos que contiene. Como resultado, el usuario tiene que buscar en dos puntos distintos para comprender el estado de cada fase, lo cual reducía la eficiencia cognitiva del tablero y afectaba la visión de conjunto.

4.1.2 Extracción de requisitos

A partir de las carencias detectadas en la interfaz original, se definen una serie de requisitos que sirven de base para el rediseño e implementación de la nueva versión del tablero Kanban:

Requisitos funcionales

- **RF01:** El sistema deberá representar cada orden de fabricación en una tarjeta, cuya posición en el tablero indique visualmente su fase actual.
- **RF02:** Cada columna del tablero representará una fase del flujo seleccionado, y las órdenes se distribuirán en función de la fase en la que se encuentren.
- **RF03:** Las tarjetas deberán mostrar información clave como el código, el nombre del cliente, la descripción y etiquetas asociadas.
- **RF04:** El sistema deberá indicar la cantidad de órdenes de fabricación que hay en una determinada fase mediante un contador visible.
- **RF05:** Los botones de acción deberán ser accesibles desde la interfaz principal del tablero Kanban.
- **RF06:** El botón de información presente en cada tarjeta deberá ser sustituido por un botón que permita acceder directamente a la edición de la orden correspondiente.

Requisitos no funcionales

- **RNF01:** La interfaz deberá facilitar la lectura visual en bloque, mediante una separación clara entre columnas.
- **RNF02:** Las tarjetas deberán tener un tamaño adecuado que permita mostrar información sin truncamientos innecesarios.
- **RNF03:** Se deberán utilizar elementos visuales diferenciadores (color, jerarquía, tamaño) para distinguir distintos tipos de órdenes.
- **RNF04:** El diseño deberá mejorar la eficiencia cognitiva del usuario, facilitando la identificación rápida de estados y elementos.
- **RNF05:** Los contadores de órdenes deberán estar alineados con el encabezado de cada columna, respetando el principio de proximidad.
- **RNF06:** El código identificador de cada orden de fabricación deberá mostrarse en negrita dentro de su tarjeta correspondiente, con el objetivo de facilitar su rápida identificación visual por parte del usuario.
- **RNF07:** La interfaz deberá presentar un diseño moderno y actualizado, acorde a los estándares actuales de usabilidad y experiencia de usuario.

4.1.3 Diseño

Una vez identificados los problemas de la interfaz original y definidos los requisitos necesarios para mejorar la experiencia de uso, se procedió a la elaboración de un nuevo diseño mediante un mockup en Figma. Este se muestra en la Figura 12.

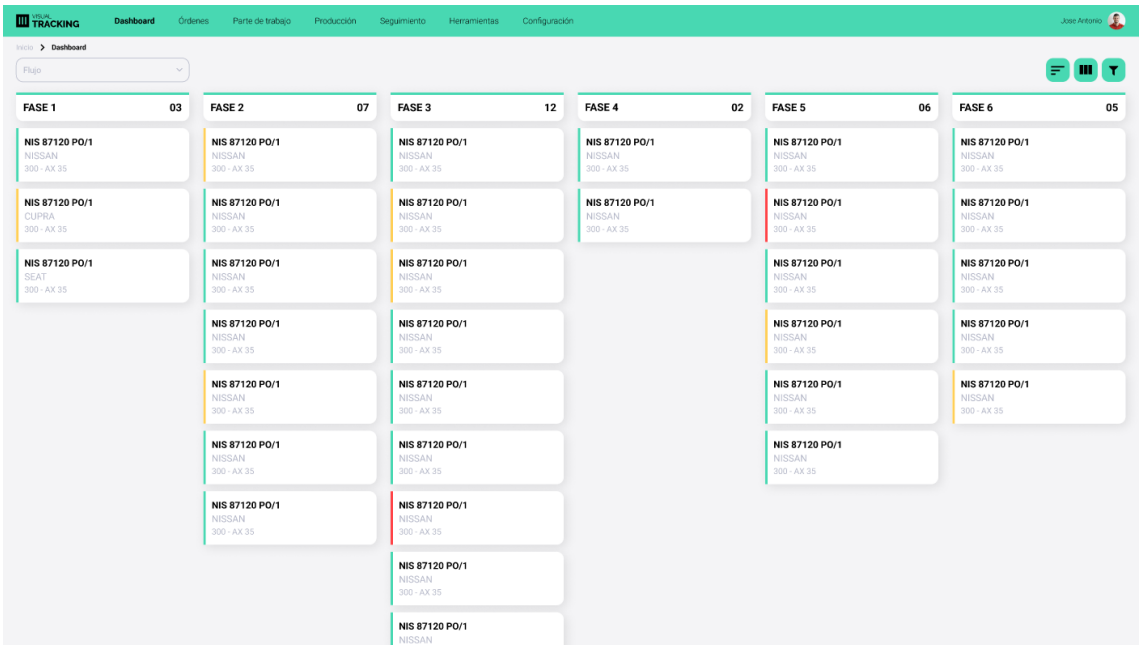

Figura 12: Diseño mockup de la interfaz Kanban a ser modificada en Figma.

4.1.3.1 Cambios realizados

Se sustituyó el color verde del fondo por un tono neutro claro. El principal objetivo de este cambio es evitar que el fondo compitiera visualmente con los elementos principales (las órdenes de fabricación). Según el principio de figura-fondo es fundamental que el fondo actúe como soporte pasivo y no desvíe la atención del contenido clave. Esta mejora facilita que el usuario mantenga el foco en las órdenes de trabajo del flujo de trabajo.

Se incorporó un fondo diferenciado con bordes redondeados a los botones situados en la parte superior derecha. Esta mejora responde a la heurística de Nielsen de “visibilidad del estado del sistema”, ya que al hacer más visibles los botones, se refuerza su carácter interactivo y se mejora su reconocimiento. Además, así, se consigue que el usuario entienda que puede hacer clic sobre ellos.

El selector de flujo se ha desplazado de la parte superior derecha a la izquierda, separándolo del grupo de botones. Esta decisión mejora la consistencia espacial y refuerza el principio de proximidad, ya que ahora los elementos con funciones diferentes están bien diferenciados. También se establece un tamaño fijo para el selector de flujo, evitando solapamientos y problemas de diseño al eliminar el tamaño variable en función del nombre del flujo.

Se ha combinado el nombre de cada fase con el contador de elementos, colocándolos juntos en la parte superior de cada columna. Esto responde al principio de proximidad de la Gestalt, que indica que los elementos relacionados deben estar agrupados visualmente. También respeta la regla de Shneiderman de reducir la carga de memoria a corto plazo, al evitar que el usuario tenga que buscar en distintos lugares de la interfaz para entender una misma unidad de información.

Se añadió una franja de color en el borde izquierdo de cada tarjeta para representar la prioridad de las órdenes (rojo = alta, amarillo = media, verde = baja). Esta solución permite que el usuario identifique rápidamente los elementos críticos, sin necesidad de leer, diferenciando unas órdenes de otras. Está respaldado por la heurística de "reconocer antes que recordar" y el principio de reducción de carga cognitiva.

Se incorporó un nuevo botón que permite ordenar automáticamente las tarjetas dentro de cada fase en función de su prioridad. Esto apoya el principio de flexibilidad y eficiencia de uso, ofreciendo a usuarios más avanzados herramientas que les permiten gestionar el flujo de forma rápida. Además, promueve la optimización del flujo de trabajo, al facilitar que las tareas más urgentes estén más visibles.

4.1.4 Implementación

Reorganización visual: El primer paso de la implementación consistió en reorganizar visualmente el tablero Kanban para mejorar la legibilidad y la jerarquía de la información. Se mantuvo el número de columnas correspondiente a las fases del flujo, pero se ajustaron el espaciado, la alineación y el tamaño relativo de los elementos mediante estilos personalizados implementados con CSS.

Además, se eliminaron los estilos visuales antiguos, como el fondo verde predeterminado, que resultaban demasiado dominantes y dificultaban la lectura prolongada. En su lugar, se aplicó un fondo neutro claro que refuerza la neutralidad visual y reduce la carga cognitiva.

Reubicación del selector de flujo y mejora de los botones: Para mejorar la agrupación lógica y reforzar la claridad funcional, se decidió reposicionar el selector de flujo a la izquierda, separándolo de los botones de filtrado, orden y visualización, los cuales permanecen alineados a la derecha.

Para los botones se definió un estilo personalizado utilizando *makeStyles* de Material UI, lo que permitió aplicar una estética coherente con el tema general de la aplicación y darles un aspecto más moderno e intuitivo.

En la Figura 13 se muestra el código empleado para definir dicho estilo, incluyendo propiedades como la redondez de los bordes, el color, el fondo y el comportamiento al pasar el cursor.

```
iconButton: {
    borderRadius: '30%',
    color: '#000000',
    backgroundColor: theme.palette.primary.main,
    '&:hover': {
        backgroundColor: theme.palette.primary.dark,
        color: theme.palette.primary.contrastText,
    },
},
```

Figura 13: Estilo aplicado a los botones de la interfaz Kanban.

Botón de ordenación por prioridad: Con el objetivo de facilitar la gestión de los ítems más relevantes dentro del tablero Kanban, se añadió un nuevo botón que permite ordenar los ítems por prioridad dentro de cada columna. Esta funcionalidad permite a los usuarios identificar rápidamente los elementos más urgentes o importantes según su nivel de prioridad asignado.

Para su implementación, se diseñó un nuevo componente reutilizable *MouseIconButton* (ver código en Figura 14). Este componente encapsula un botón con *tooltip,* al que se le añade un icono *<Sort />* de Material UI que refuerza la intuición de su funcionaliad.

```
export default function MouseIconButton(props) {
    const {
        title,
        children,
        onClick,
        popperClasses,
        active = false,
    } = props;

    const classes = useStyles();

    return (
        <Tooltip title={title}>
            <IconButton
                className={`${classes.iconButton} ${active ? classes.active : ''}`}
                onClick={onClick}
                classes={popperClasses}
            >
                {children}
            </IconButton>
        </Tooltip>
    );
}
```

Figura 14: Componente MouseIconButton.

En cuanto a la lógica, al activar este botón se modifica el estado *sortByPriorit.* Durante el renderizado de las columnas, esta variable de estado se utiliza para reordenar las tarjetas (*cards*) aplicando un algoritmo de ordenación descendente por prioridad, tal como se muestra en la Figura 15.

```
const [sortByPriority, setSortByPriority] = React.useState(false);

    if (sortByPriority) {
        cards = [...cards].sort((a, b) => (b.priorityId ?? 0) - (a.priorityId ?? 0));
    }
```

Figura 15: Lógica del botón ordenar por prioridad.

El botón que activa esta funcionalidad se diseñó con un comportamiento visual claro para el usuario. En estado inicial, se muestra con un aspecto neutro y un *tooltip* informativo con el texto “Ordenar por prioridad”, como se puede ver en la Figura 16.

La Figura 17 muestra cómo una vez pulsado, el botón cambia de aspecto para indicar que la ordenación está activa, y el *tooltip* se actualiza a “Ordenado por prioridad”, proporcionando retroalimentación visual inmediata sobre el estado actual de la interfaz.

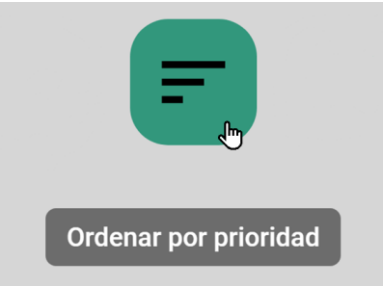

Figura 16: Botón ordenar por prioridad sin pulsar.

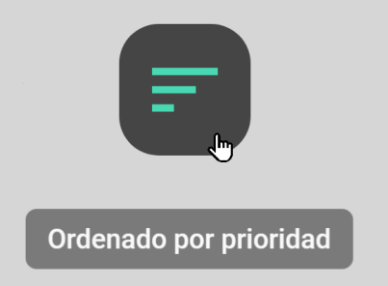

Figura 17: Botón ordenar por prioridad activo.

Nombre de la fase y contador de órdenes: Cada columna del tablero Kanban representa una etapa del flujo de trabajo, por lo que su encabezado debe proporcionar información clara y resumida sobre el estado correspondiente.

Para ello, se diseñó un título de columna que combina el nombre de la etapa junto con el contador de ítems, mostrando cuántas órdenes se encuentran actualmente en esa etapa. Esto permite a los usuarios tener una visión rápida del volumen de trabajo en cada fase del flujo.

A nivel de implementación, se utilizó un contenedor * * que organiza dos elementos ** alineados horizontalmente mediante *flexbox*, distribuyéndolos en los extremos: el nombre de la etapa (*step.name*) a la izquierda y el número de tarjetas (*cards.length*) a la derecha. El contador se formatea con dos dígitos utilizando *padStart*, garantizando uniformidad visual en todas las columnas, independientemente del número de ítems. En la Figura 18 se muestra una captura del código empleado.

```
title: (
    <div style={{ display: 'flex', justifyContent: 'space-between', width: '100%' }}>
        <span>{step.name}</span>
        <span>{cards.length.toString().padStart(2, '0')}</span>
    </div>
),
```

Figura 18: Nombre y contador de cada fase.

Prioridades en las órdenes de fabricación: Con el fin de reforzar la legibilidad y priorización visual de las órdenes de fabricación dentro del tablero Kanban, se implementó un sistema de codificación por colores basado en el nivel de prioridad de cada ítem. Esta mejora tiene como objetivo principal ayudar a los usuarios a detectar rápidamente aquellas tareas que requieren mayor atención o intervención.

Cada tarjeta (orden de fabricación) cuenta ahora con un indicador de color en el borde izquierdo, cuya tonalidad varía según el nivel de prioridad asignado al ítem:

- Prioridad baja: borde de color verde (#48D8B2).
- Prioridad media: borde de color amarillo (#F0AD4E).
- Prioridad alta: borde de color rojo (#D9534F).

Este sistema visual sigue el patrón estándar utilizado en contextos industriales para representar niveles de urgencia o riesgo, facilitando su comprensión incluso por parte de usuarios poco familiarizados con la herramienta.

Para implementar esta funcionalidad, se definió un objeto *borderColor* que asocia cada *priorityId* con un color específico predefinido en la constante *COLORS*. Posteriormente, este color se aplica de forma dinámica utilizando la propiedad *style* del contenedor principal de la tarjeta para modificar su borde izquierdo (*borderLeft*) como se puede ver en la Figura 19.

```
const borderColor = {
    1: COLORS.LowPriority,
    2: COLORS.MediumPriority,
    3: COLORS.HighPriority,
}[card.priorityId] || COLORS.LowPriority;

<div
    className={cardClasses}
    onClick={() => openEditModal(card)}
    style={{ textDecoration: 'none', color: 'inherit', borderLeft: `5px solid ${borderColor}` }}
>
```

Figura 19: Aplicación del color de prioridad mediante borde izquierdo en las tarjetas del tablero Kanban.

Editar una orden de fabricación: Con el objetivo de mejorar la usabilidad de la interfaz en entornos industriales, se sustituyó el botón de información (representado previamente por un icono “i”) por un nuevo botón de interacción más accesible y ergonómico.

En lugar de requerir que el usuario pulse un botón específico y de pequeñas dimensiones, se optó por hacer que toda la tarjeta del tablero Kanban actuara como área interactiva. Al pulsar en cualquier parte de la tarjeta, se abre directamente el formulario de edición de la orden correspondiente.

Esta decisión de diseño responde a la necesidad de facilitar el uso de la aplicación en condiciones donde los usuarios pueden estar utilizando guantes, lo que

dificulta interacciones precisas sobre elementos pequeños. Al ampliar la superficie de pulsación, se reduce considerablemente la probabilidad de errores de interacción y se mejora la eficiencia operativa.

Técnicamente, esta funcionalidad se implementó gestionando el evento *onClick* sobre el contenedor principal de cada tarjeta. Este evento invoca la función *openEditModal(card)*, la cual se encarga de iniciar el flujo de edición. En su interior, dicha función actualiza el contexto global mediante un *dispatch* con el tipo *'SELECT_ITEM'* y el ítem seleccionado como *payload*, almacenando así la tarjeta actual en el estado compartido de la aplicación. A continuación, se ejecuta *setIsEditModalOpen(true)* para mostrar el componente modal que contiene el formulario de edición (ver Figura 20). Esta lógica se puede ver con detalle en la Figura 21

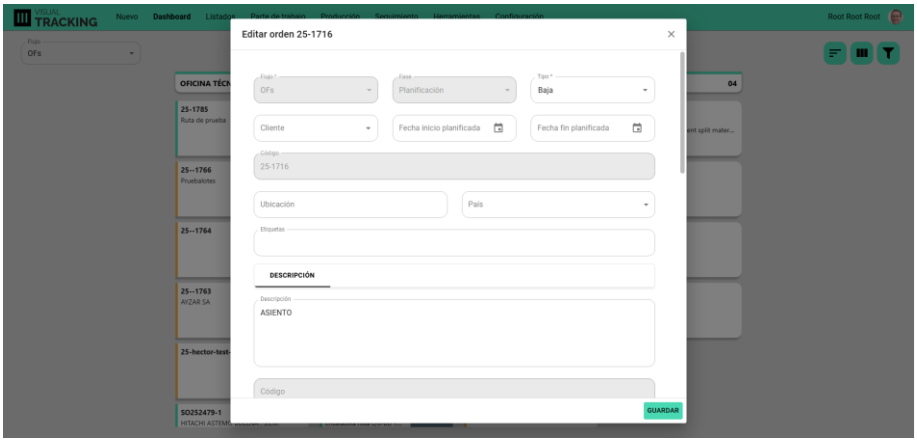

Figura 20: Formulario de edición de una orden de fabricación.

```jsx
const [openEditDialog, setOpenEditDialog] = React.useState(false);

const openEditModal = card => {
    setSelectedItem(card);
    dispatch(selectItem(card));
    setOpenEditDialog(true);
}

{openEditDialog && selectedItem && (
    <ItemTrackingProvider
        values={{
            flow: selectedFlow,
            step: stepsById[selectedItem?.stepId] ?? null,
            item: selectedItem,
        }}
    >
        <EditItem
          open={openEditDialog}
          setOpen={setOpenEditDialog}
          title={i18n.formatString(i18n.editItem, selectedItem.code)}
          itemId={selectedItem.id}
          onSaved={handleSaved}
      />
    </ItemTrackingProvider>
)}

const RenderCard = props => {
  ...
  return (
      <div
          className={cardClasses}
          onClick={() => openEditModal(card)} // ← Toda la tarjeta es clicable
          style={{ textDecoration: 'none', color: 'inherit', borderLeft: `5px solid ${borderColor}` }}
      >
          ...
      </div>
  );
}
```

Figura 21: Activación del formulario de edición mediante clic en toda la tarjeta.

Visualización de imágenes en las órdenes de fabricación: Una vez finalizada la interfaz Kanban de las órdenes de fabricación basada en el diseño previamente elaborado en Figma, mi tutor en la empresa solicitó una mejora adicional. Esta solicitud dio lugar a un nuevo requisito funcional: incluir una imagen en la tarjeta que muestra la información principal de cada orden de fabricación, siempre que dicha orden disponga de documentación gráfica asociada.

Para llevar a cabo esta funcionalidad, fue necesario recuperar dos tipos de datos desde el *backend.*

1. Metadatos de las imágenes: A partir del listado de órdenes de fabricación disponibles (*items.results*), se genera una consulta al *endpoint vt.itemPicture.searchAll*, filtrando por los identificadores de ítems (ver código en Figura 22). Esta primera consulta recupera los metadatos de las imágenes, como el identificador del archivo adjunto (*attachmentUuid*), el ID del ítem, si la imagen es principal, y otros campos relevantes.

```
const itemPicturesQueryKey = vt.itemPicture.searchAll({
    size: 1000,
    values: {
        field: 'ITEM_ID',
        operator: FILTER_OPERATORS.IN,
        values: items?.results?.map(item => item.id),
    },
});
```

Figura 22: Petición a la API para obtener los metadatos de las imágenes.

2. Contenido de las imágenes: Con los *attachmentUuid* obtenidos en la primera consulta, se lanza una segunda petición al *endpoint documentsApi.legacyFile.searchAll*, esta vez solicitando los datos binarios (*fileData*) de cada imagen (ver código en Figura 23). El uso de *includeData*: *true* en el contenido de la query permite que se devuelvan los archivos codificados en Base64 junto con metadatos como el tipo MIME o el nombre del archivo.

```
const documentsSearchQueryKey = documentsApi.legacyFile.searchAll({
    content: {
        ids: picturesData?.content?.map(picture => picture?.attachmentUuid),
        includeData: true,
    },
});
```

Figura 23: Petición a la api de los datos relacionados con las imágenes.

A partir de esto, se asocian los datos obtenidos en ambas consultas, combinando los metadatos de *picturesData* con la información detallada del archivo (*fileData, name, mimeType*) proveniente de *documentsSearch* como se muestra en la

Figura 24. El resultado es un array de objetos que contiene toda la información necesaria para mostrar correctamente las imágenes asociadas a cada ítem en la interfaz de usuario.

```
const pictures = React.useMemo(() => {
    return (picturesData?.content || []).map(document => {
        const file = documentsSearch?.results.find(item => item.id === document.attachmentUuid);
        return file ? { ...document, fileExt: file?.mimeType, fileName: file?.name, fileData: file?.fileData } : document;
    });
}, [documentsSearch?.results, picturesData]);
```

Figura 24: Composición del array con los metadatos y datos binarios de las imágenes.

Finalmente, en el componente visual encargado de renderizar las tarjetas de cada orden (*RenderCard*), se filtran las imágenes disponibles para obtener solo aquellas que están vinculadas al ítem actual (*card.id)*. Si existe una única imagen, será la que se mostrará en la interfaz. Si hay varias, se busca la imagen principal filtrando por el campo *principal.* Si no se encuentra ninguna principal, se selecciona la primera imagen disponible.

Para renderizar la imagen, esta se muestra dentro de un *div* usando el componente Avatar de Material-UI. Este componente permite renderizar directamente una imagen codificada en Base64, como se muestra en la Figura 25.

```
const RenderCard = props => {
    const { selectedItem, card, people, openEditModal, pictures } = props;

    const itemPictures = pictures.filter(picture => picture.itemId === card.id);
    const picture = itemPictures.length === 1 ? itemPictures[0] : itemPictures.find(picture => picture.principal) ||
        itemPictures[0];

    ...

    {picture && (
       <div className="item-image">
           <Avatar
               src={`data:${picture.fileExt};base64,${picture.fileData}`}
               variant='circular'
           />
       </div>
    )}
}
```

Figura 25: Búsqueda y renderizado de la imagen correspondiente a una orden.

En cuanto a la estructura de la tarjeta, esta está dividida en dos bloques principales. Primero el código del ítem (Figura 26), se presenta una línea superior que ocupa el 100% del ancho de la tarjeta. Esto responde a la necesidad de evitar truncamientos en el código, ya que se trata del dato más relevante para la identificación rápida de la orden.

```
<div className="col-12 pr-0 item-code">{card.code}</div>
```

Figura 26: Código para el renderizado del código identificador del ítem.

Debajo del código, se agrupa el resto de los datos en un contenedor con diseño *flex* en dirección horizontal, que se divide en dos bloques:

- Bloque de datos *(item-data*), el cual se sitúa a la izquierda y contiene el nombre del cliente (*item-client*), la descripción del ítem (*item-description*) y las etiquetas asociadas (*OfTags*). Este bloque ocupa el 80% del ancho del contenedor de datos si hay una imagen disponible. En caso contrario, si no hay imágenes que mostrar, se amplía su ancho hasta el 95% gracias a la clase *full-width* de CSS como se muestra en la Figura 27.

```
<div className={classNames("item-data", { "full-width": !picture })}>
    <div className="pr-0 h-100 item-client">
        {people?.find(person => person.id === card.personId)?.name}
    </div>
    <div className="pr-0 item-description">{card.description}</div>
    <div className="pr-0 item-description pt-1">
        <OfTags tags={card.tags ?? []} />
    </div>
</div>

.item-data {
    flex: 0 0 80%;
    max-width: 80%;
}

.full-width {
    flex: 0 0 95%;
    max-width: 95%;
}
```

Figura 27: Código para el renderizado de los datos de la orden.

- Bloque de imagen (*item-image*). Este bloque existe únicamente si hay por lo menos una imagen asociada al ítem. En ese caso, se muestra un Avatar con forma circular, alineado al borde inferior derecho de la tarjeta. Su tamaño se ajusta automáticamente con un *max-width* del 20%, conservando el equilibrio visual sin desplazar el contenido textual (ver código en Figura 28):

```
{picture && (
    <div className="item-image">
        <Avatar
            src={`data:${picture.fileExt};base64,${picture.fileData}`}
            variant='circular'
        />
    </div>
)}

 .item-image {
    width: auto;
    max-width: 20%;
    align-self: end;
    margin-right: 5px;
    margin-bottom: 5px;
}
```

Figura 28: Código para el renderizado de la imagen de una orden.

La Figura 29 muestra el código HTML y CSS empleado para representar cada orden de fabricación como una tarjeta dentro del tablero Kanban.

```jsx
<div className="row no-gutters" style={{ padding: '0.4rem 0 0 0.4rem', width: '100%'}}>
    <div className="col-12 pr-0 item-code">{card.code}</div>
    <div className="item-data-container">
        <div className={classNames("item-data", { "full-width": !picture })}>
            <div className="pr-0 h-100 item-client">
                {people?.find(person => person.id === card.personId)?.name}
            </div>
            <div className="pr-0 item-description">{card.description}</div>
            <div className="pr-0 item-description pt-1">
                <OfTags tags={card.tags ?? []} />
            </div>
        </div>
        {picture && (
            <div className="item-image">
                <Avatar
                    src={`data:${picture.fileExt};base64,${picture.fileData}`}
                    variant='circular'
                />
            </div>
        )}
    </div>
</div>

.item-data-container {
    width: 100%;
    height: 87px;
    display: flex;
    flex-direction: row;
    align-items: flex-start;
    justify-content: space-between;
}
```

Figura 29: Código para el renderizado de las tarjetas del tablero Kanban.

La Figura 30 muestra el resultado final de la interfaz. Este nuevo diseño permite una presentación visual limpia, jerárquica y adaptativa. El código del ítem se mantiene completamente visible en la parte superior, seguido por una distribución eficiente del contenido: los datos clave en la parte izquierda y la imagen (si la hay) a la derecha, ocupando un espacio reducido pero visualmente destacado. Además, el diseño se adapta correctamente en el caso de no existir ninguna imagen, aprovechando al máximo el espacio disponible para mostrar los datos de forma clara.

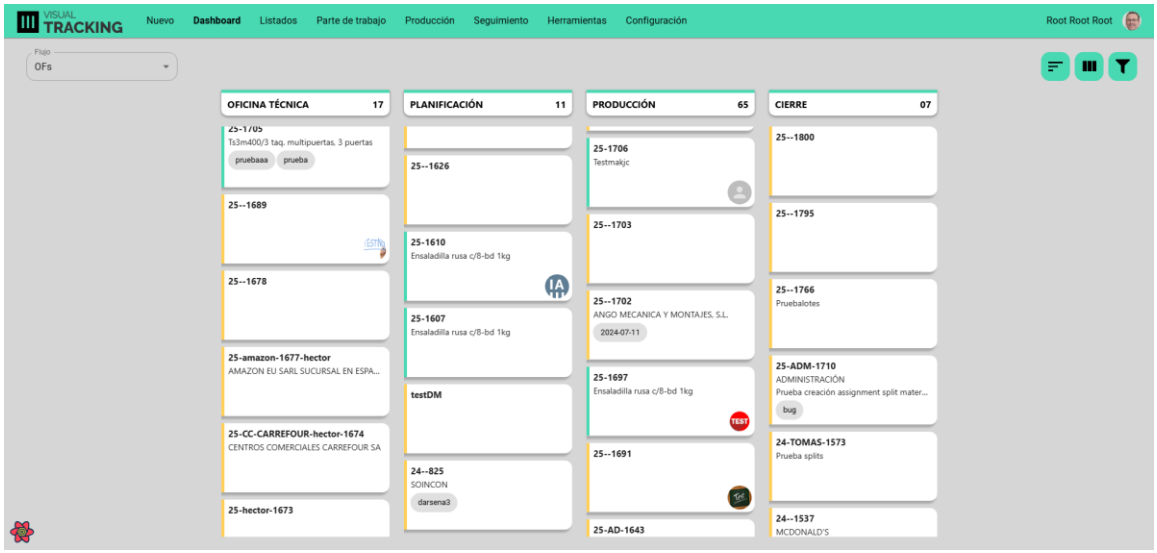

Figura 30: Interfaz Kanban rediseñada.

4.2 Secuenciador

El secuenciador de órdenes de fabricación (ver Figura 31) es una herramienta fundamental para la planificación y programación a capacidad finita de las actividades de producción. Su principal objetivo es permitir una visualización clara y controlada de las actividades productivas en un diagrama de Gantt, teniendo en cuenta múltiples restricciones del entorno industrial, como la disponibilidad de operarios, materiales en stock, utillajes, plazos de entrega, estado de las máquinas y capacidad de los recursos. Esta segunda interfaz también forma parte del módulo Visual Tracking dentro de EMI Suite 4.0 y se complementa con la vista Kanban previamente presentada.

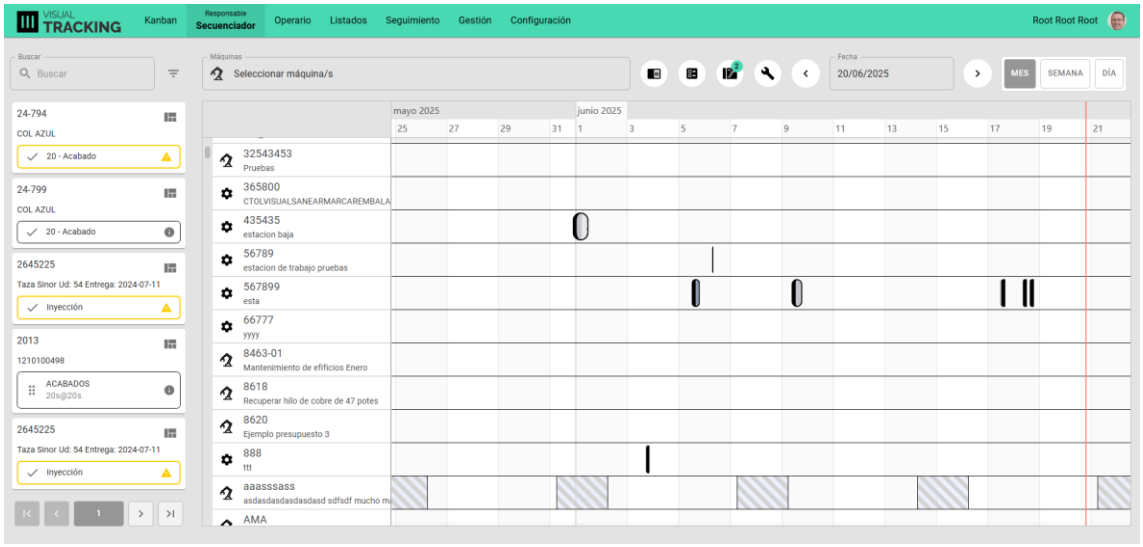

Figura 31: Interfaz original del secuenciador de órdenes.

4.2.1 Análisis de la interfaz original

La interfaz del secuenciador se presenta como un diagrama de Gantt que representa la programación de las órdenes de fabricación. El eje de abscisas corresponde a la línea temporal, y el eje de coordenadas muestra el listado de máquinas o estaciones de trabajo disponibles en planta. A través de esta visualización, el usuario puede consultar de forma rápida qué órdenes están asignadas a cada máquina, en qué franja horaria y con qué duración.

En la parte izquierda de la interfaz se muestra un panel de búsqueda que permite localizar órdenes disponibles, las cuales pueden arrastrarse hacia el diagrama para su programación manual. En la parte superior se encuentran herramientas de filtrado por máquinas, selección de la fecha actual y escala temporal (día, semana o mes) así como controles para aumentar o disminuir el zoom, limpiar los filtros etc.

Cada orden asignada aparece representada como un bloque rectangular dentro de la fila correspondiente a una máquina. Estos bloques incluyen la duración estimada de la tarea, el código o identificador y una pequeña imagen en el caso de disponer de ella.

Aunque la interfaz cumple correctamente su función a nivel visual y de usabilidad, en su uso real se identifican problemas de rendimiento que afectan negativamente a la experiencia del usuario. Uno de los principales inconvenientes aparece al desplazarse horizontalmente por el eje temporal (*scroll* lateral) o al reducir el nivel de zoom para visualizar un mayor rango de fechas. En estos casos y como se muestra en la Figura 32, se produce una recarga visible y constante de la interfaz. Este comportamiento no solo provoca una experiencia de usuario interrumpida y poco fluida, sino que puede resultar molesta y, en ciertos casos, frustrante para el usuario.

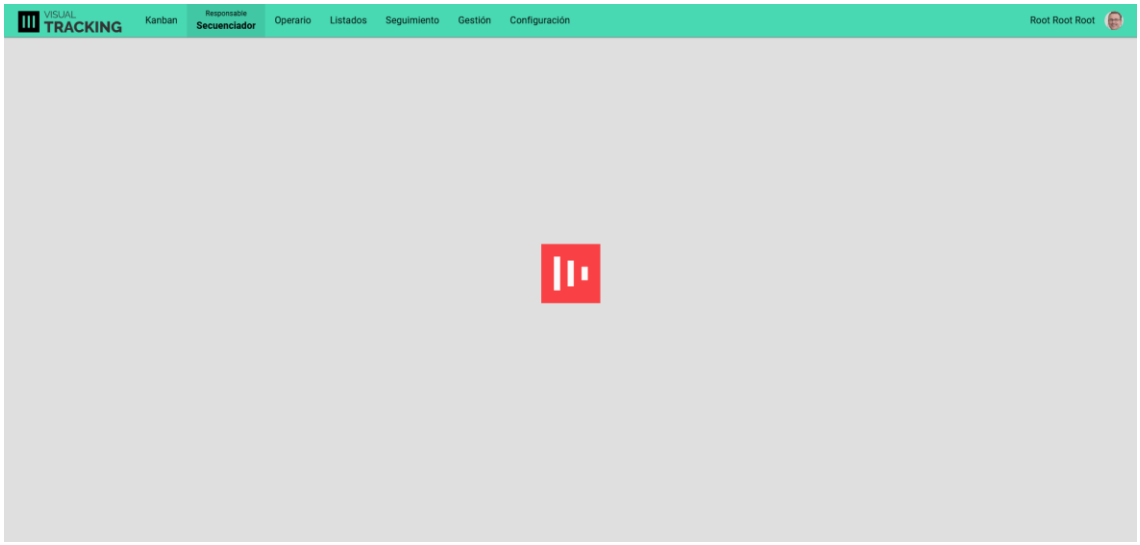

Figura 32: Pantalla de carga de EMI Suite 4.0.

Además, el rendimiento se ve aún más comprometido en aquellos casos en los que existe un elevado número de órdenes representadas simultáneamente. A medida que aumenta la carga visual, la interfaz comienza a mostrar signos de lentitud, como retrasos en el renderizado, bloqueos momentáneos o pérdida de respuesta al intentar interactuar con las órdenes.

Ante estas limitaciones, se optó por mantener la apariencia visual de la interfaz original, intentando lograr un resultado lo más idéntico posible en cuanto a diseño y funcionalidad y, con el objetivo de mejorar el rendimiento, me propusieron reconstruir el componente del secuenciador utilizando la librería Highcharts. La idea principal de esto era comprobar si la utilización de dicha librería podía mejorar el rendimiento general del componente. Esta decisión no aseguraba de antemano una solución final, sino que respondía a un enfoque experimental, motivado por el hecho de que Highcharts ya estaba siendo utilizada en otros módulos de la aplicación para representar planificaciones similares. De este modo, se buscaba no solo una posible mejora de rendimiento, sino también una mayor coherencia dentro del sistema.

4.2.2 Extracción de requisitos

A partir de las limitaciones detectadas en la versión original del secuenciador, especialmente en lo relativo al rendimiento y la experiencia de uso, se definieron una serie de requisitos para guiar el desarrollo de una nueva implementación que mantuviera el diseño original, pero mejorara su comportamiento.

Requisitos funcionales

- **RF01:** El sistema deberá mostrar un diagrama de Gantt en el que el eje horizontal represente el tiempo y el eje vertical las distintas máquinas disponibles para la producción.

- **RF02:** La representación gráfica de las órdenes de fabricación deberá realizarse reutilizando el mismo componente que ya existe en la interfaz original, de manera que se mantenga la organización visual y funcional previamente establecida.

- **RF03:** El usuario deberá poder desplazarse horizontalmente a lo largo de la línea temporal manteniendo pulsado el botón izquierdo del ratón y arrastrándolo lateralmente. Asimismo, deberá poder desplazarse verticalmente utilizando la rueda del ratón y aplicar zoom combinando la tecla Control con la rueda del ratón.

- **RF04:** La nueva implementación del secuenciador deberá ser compatible con el panel superior de filtros, de forma que se actualice su contenido en función de los criterios seleccionados por el usuario.

- **RF05:** Las órdenes deberán agruparse por máquina, respetando el orden y la estructura establecida en la versión original del secuenciador.

- **RF06:** Las órdenes de fabricación deberán poder ser reprogramadas mediante la funcionalidad de *drag and drop*, permitiendo su desplazamiento directo dentro del diagrama de Gantt.

- **RF07:** En caso de que una operación se solape con otra durante una reprogramación, el sistema deberá detectar el conflicto y mostrar un cuadro de diálogo informando al usuario del problema, así como ofrecer opciones para resolverlo.

- **RF08:** Los periodos en los que una máquina no esté disponible deberán representarse visualmente mediante bloques con fondo transparente y líneas diagonales, indicando claramente que en ese intervalo de tiempo no se pueden planificar operaciones.

Requisitos no funcionales

- **RNF01:** Se deberá utilizar la librería Highcharts para el desarrollo del nuevo secuenciador.

- **RNF02:** El diseño visual de la nueva interfaz deberá replicar fielmente la estética y disposición del secuenciador original, de modo que el usuario no perciba diferencias significativas en la experiencia visual ni en la disposición de los elementos.

- **RNF03:** La interfaz deberá ofrecer una respuesta rápida y fluida ante acciones del usuario, incluso en escenarios con un alto número de órdenes de fabricación visualizadas simultáneamente.

4.2.3 Implementación

La implementación del nuevo secuenciador se ha llevado a cabo reutilizando la estructura de componentes ya existente en la aplicación, introduciendo como núcleo funcional una nueva clase denominada GanttChart, basada en la librería Highcharts Gantt. El objetivo principal ha sido mantener la apariencia visual original y mejorar el rendimiento en la visualización de órdenes de fabricación.

El componente principal del secuenciador es *SequencerUI*, una clase de alto nivel que encapsula el contexto necesario para la gestión de estado global utilizando React Context API. Esta clase organiza la composición de los distintos proveedores (*UIProvider, SequenceProvider y FilterSequencerProvider*) y renderiza la interfaz principal del secuenciador como se ilustra en la Figura 33.

```
export default function SequencerUI() {
    return (
        <UIProvider>
            <SequenceProvider>
                <FilterSequencerProvider>
                    <Sequencer />
                </FilterSequencerProvider>
            </SequenceProvider>
        </UIProvider>
    );
}
```

Figura 33: Estructura del componente *SequencerUI* con proveedores de contexto.

Dentro de *SequencerUI*, se encuentra el componente *Sequencer*, que se encarga de organizar la disposición visual de la interfaz. Este incluye otros componentes como:

- *SearchItemns*: panel lateral para buscar órdenes.

- *FilterOptions*: panel superior para aplicar los filtros de visualización.
- *GanttChart*: el componente encargado de renderizar el diagrama de Gantt mediante HighchartsReact.
- *SequencerModal*: un modal contextual para gestionar conflictos o editar órdenes.

El componente clave aquí es *GanttChart*, que encapsula toda la lógica relacionada con Highcharts, concretamente con el módulo Highcharts Gantt. Este componente se diseñó como un componente funcional de React, optimizado mediante *React.memo* para evitar renderizados innecesarios. Hace uso del contexto *useSequenceProvider* que gestiona todos los datos y funciones globales de la aplicación.

Una de las primeras tareas en la implementación del componente fue transformar los datos de secuenciación y máquinas en el formato esperado por la librería. Highcharts requiere que el eje de coordenadas se defina como una lista de categorías. En este caso, dichas categorías corresponden al nombre de cada máquina (*machine.name*). Para ello, y como se muestra en la Figura 34, se extrajo la lista ordenada de nombres desde el objeto *machinesData*, esta lista se usa posteriormente para determinar la posición vertical (eje de coordenadas) de cada orden en el diagrama.

```
// Get the machine categories from the machinesData
const machineCategories = useMemo(() => machinesData?.content.map(machine => machine.name) || [], [machinesData]);
```

Figura 34: Ejemplo de formato de datos para eje Y con nombres de máquina.

A continuación, se prepararon los datos de las órdenes (*sequencerData.sequencedSplits*) como se muestra en la Figura 35.

```
// Prepare the data for the chart
const preparedData = useMemo(() => {
    if (!sequencerData || !machinesData?.content) return [];

    return sequencerData.sequencedSplits
        .map(split => {
            const start = moment(split.assignmentSplit.startDateTime).valueOf();
            const end = moment(start)
                .add(split.assignmentSplit.duration, 'seconds')
                .valueOf();

            const machineName =
                machinesData.content.find(machine => machine.id === split.assignmentSplit.workstationId)?.name || 'Desconocida';

            return {
                id: `task-${split.assignmentSplit.id}`,
                name: split.item.code,
                start,
                end,
                custom: { item: split.item, splitId: split.assignmentSplit.id, split: split },
                y: machineCategories.indexOf(machineName),
            };
        })
        .sort((a, b) => a.start - b.start);
}, [sequencerData, machinesData, machineCategories]);
```

Figura 35: Función que transforma los datos de las órdenes a una estructura compatible con Highcharts.

Esta función realiza tres operaciones:

1. Conversión de fechas: Utiliza *Moment.js* para obtener el valor en milisegundos de inicio y fin de cada tarea.

2. Asociación con máquina: Busca el nombre de la máquina correspondiente a la *workstationId* del *split*, y calcula su índice en *machineCategories* para ubicarla en el eje Y.
3. Definición de la estructura del objeto: Cada entrada incluye: un identificador único para Highcharts, el código del ítem asociado a la tarea, valores de comienzo y fin en formato *timestamp*, un objeto adicional que guarda información útil para interacciones posteriores y el índice de la máquina en el eje vertical.

Además de las tareas productivas, el componente Gantt también muestra las franjas de tiempo en las que una máquina no está disponible. Como se muestra en la Figura 36, se recorre la lista de máquinas y se busca, para cada una, sus franjas horarias. Luego, mediante la función *getNonProductiveSlots*, se filtran aquellos espacios de tiempo en los que la máquina no está activa y se adaptan al formato requerido por el gráfico. Cada franja se representa con *su start, end* y la posición en el eje y correspondiente a la máquina.

```
// Get non-productive slots for the chart
const nonProductiveSlots = useMemo(() => {
    const workstationsIds = removeDuplicates((machinesData?.content ?? []).map(machine => machine.id));
    const timelineDateStart = moment(sequencerStartDate ?? datesRef.current.start);
    const timelineDateEnd = moment(sequencerEndDate ?? datesRef.current.end);

    return workstationsIds
        .map(workstationId => {
            const machineTimeSlots = timeSlots?.find(ts => ts.assetId === workstationId)?.timeSlots;

            return machineTimeSlots
                ? getNonProductiveSlots({
                      workstationId,
                      machineTimeSlots,
                      timelineDates: { start: timelineDateStart, end: timelineDateEnd },
                  })
                : [];
        })
        .flat()
        .map((slot, i) => ({
            id: `${NON_PRODUCTIVE_SLOTS_KEY}_${slot.workstationId}_${i}`,
            start: moment(slot.start).valueOf(),
            end: moment(slot.end).valueOf(),
            y: machineCategories.indexOf(machinesData.content.find(machine => machine.id === slot.workstationId)?.name),
        }));
}, [machinesData, timeSlots, sequencerStartDate, sequencerEndDate, datesRef, machineCategories]);
```

Figura 36: Función que transforma los datos de los slots inactivos a una estructura compatible con Highcharts.

Para la funcionalidad de zoom utilizando la rueda del ratón combinada con la tecla Ctrl, se definió una función *handleWheel* que detecta si el usuario mantiene pulsada la tecla Ctrl al hacer *scroll* tal como se muestra en la Figura 37.

```
// Zoom in/out with the mouse wheel while holding the Ctrl key
const handleWheel = useCallback(
    event => {
        if (!event.ctrlKey) return;
        event.preventDefault();

        const zoomFactor = event.deltaY < 0 ? 0.8 : 1.2;
        handleZoom(zoomFactor);
    },
    [handleZoom]
);
```

Figura 37: Función personalizada de zoom con rueda y tecla Ctrl.

La función *handleZoom* (ver Figura 38) ajusta los límites del eje X del gráfico. Calcula un nuevo rango alrededor del punto medio del eje y actualiza los extremos (min y max) del eje temporal utilizando la API de Highcharts, logrando así un efecto de zoom progresivo centrado.

```
const handleZoom = useCallback(zoomFactor => {
    const chart = chartRef.current?.chart;
    if (!chart) return;

    const axis = chart.xAxis[0];
    const range = axis.max - axis.min;
    const midPoint = (axis.max + axis.min) / 2;

    const newMin = midPoint - (range * zoomFactor) / 2;
    const newMax = midPoint + (range * zoomFactor) / 2;

    axis.setExtremes(newMin, newMax);
}, []);
```

Figura 38: Función que calcula el nuevo rango de fechas para aplicar zoom.

Para que esta interacción funcione correctamente, se utilizó un *useEffect* que accede directamente al container del gráfico y le añade un *event listener* al evento w*heel* como se aprecia en la Figura 39.

```
useEffect(() => {
    const chart = chartRef.current?.chart;
    if (!chart) return;

    const container = chart.container;
    container.addEventListener('wheel', handleWheel, { passive: false });

    return () => {
        container.removeEventListener('wheel', handleWheel);
    };
}, [chartRef, handleWheel]);
```

Figura 39: Implementación del evento *wheel* con *useEffect*.

Esta lógica personalizada fue necesaria porque el comportamiento de zoom predeterminado de Highcharts (basado en selección por arrastre o zoom automático) no se adaptaba a los requisitos de interacción específicos del proyecto.

Drag and drop: El componente permite mover tareas dentro del diagrama de Gantt mediante *drag and drop*. Cuando el ítem es soltado, es decir, ya se ha cambiado su posición se ejecuta la función *handleDrop* (ver Figura 40), la cual llama a *checkConflicts* y si hay conflictos entre las órdenes, se cancela el movimiento avisando al usuario y proporcionándole distintas soluciones, como se muestra en la Figura 41. Si por el contrario no hay conflictos, se actualiza la propiedad *startDateTime* de la orden y ejecuta la mutación *updateAssignmentSplit* para guardar el cambio.

```
const handleDrop = useCallback(
    event => {
        if (checkConflicts(event)) {
            return;
        }

        const splitId = event.target.custom.splitId;
        const assignmentSplit = sequencerData.sequencedSplits.find(s => s.assignmentSplit.id === splitId).assignmentSplit;

        const startDateTime = new Date(event.newPoint.start).toISOString();
        assignmentSplit.startDateTime = startDateTime;

        updateAssignmentSplit(assignmentSplit);
    },
    [sequencerData, updateAssignmentSplit]
);
```

Figura 40: Lógica de la función *handleDrop* al soltar una orden.

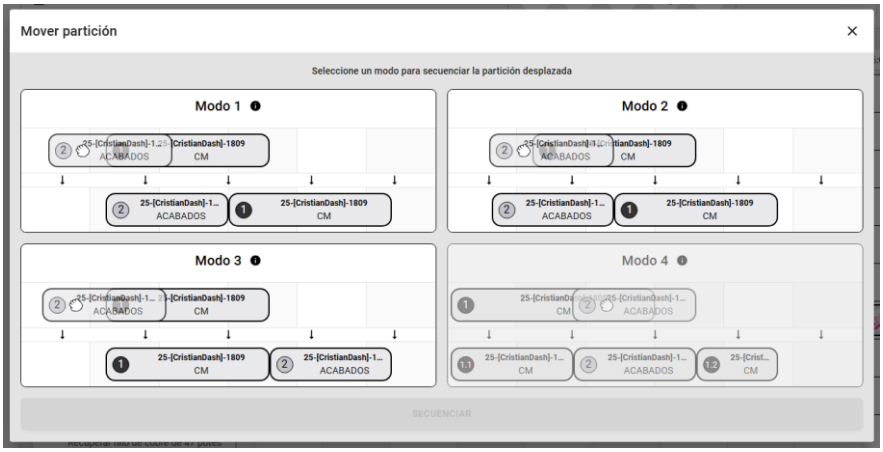

Figura 41: Modal para la gestión de conflictos al soltar una orden.

A continuación se explican las distintas opciones de configuración implementadas para el diagrama Gantt de Highcharts:

Configuración general del gráfico (ver Figura 42):

- styledMode: se usa CSS externo para los estilos.
- type: 'gantt': tipo de gráfico.
- ignoreHiddenSeries: optimización al ignorar series ocultas.
- zooming y panning: se desactiva el zoom con la rueda (porque usamos uno personalizado) y se permite hacer scroll horizontal con el ratón.

```
chart: {
    styledMode: true,
    type: 'gantt',
    ignoreHiddenSeries: true,
    zooming: {
        mouseWheel: { enabled: false },
    },
    panning: {
        enabled: true,
        type: 'x',
    },
}
```

Figura 42: Configuración general del gráfico Gantt.

Rendimiento y apariencia (ver Figura 43):

- boost: mejora el rendimiento en gráficos con muchas series usando WebGL.
- Se desactiva la marca de agua de Highcharts y el *tooltip* por defecto.

```
boost: {
    useGPUTranslations: true,
    seriesThreshold: 2,
},
credits: { enabled: false },
tooltip: { enabled: false },
```

Figura 43: Ajustes de rendimiento y apariencia del gráfico.

Opciones por serie (ver Figura 44)**:**

- turboThreshold: 0, evita que Highcharts limite el número de puntos.
- animation: sin animaciones para mejor rendimiento.
- boostThreshold: se habilita el modo *boost* para un mejor rendimiento.
- dataLabels: se usa HTML para poder insertar etiquetas personalizadas dentro de las barras del Gantt con el texto alineado a la izquierda.

```
plotOptions: {
    series: {
        turboThreshold: 0,
        animation: false,
        boostThreshold: 1,
        dataLabels: {
            useHTML: true,
            align: 'left',
            allowOverlap: true,
        },
    },
},
```

Figura 44: Opciones de configuración por serie en Highcharts.

Eje X (ver Figura 45)**:**

- type: se configura como eje temporal.
- currentDateIndicator: se muestra una línea vertical que indica la fecha actual.
- alternateGridColor: se alternan los colores del fondo entre un gris claro y blanco.

```
xAxis: {
    type: 'datetime',
    currentDateIndicator: true,
    alternateGridColor: '#fafafa',
},
```

Figura 45: Configuración del eje temporal (X).

Eje Y (ver Figura 46)**:**

- UniqueNames: evita que las máquinas aparezcan repetidas en el eje y.
- StaticScale: define la altura de cada fila.
- Categories: cada fila del eje y es una máquina o estación de trabajo.
- En las etiquetas *(labels.formatter*), se renderiza un icono junto con el código y el nombre de cada máquina, usando HTML y *ReactDOMServer* para representar cada máquina (ver Figura 47).

```
yAxis: {
    uniqueNames: true,
    staticScale: 56, // Y-axis height
    categories: machineCategories,
    labels: {
        useHTML: true,
        formatter: function() {
            const machine = machinesData?.content?.find(m => m.name === this.value);
            if (!machine) return this.value;

            const icon = getAssetIcon(assetsTypesGeneric[machine.type] || '');

            return `
            <div class="machine-data-container">
                <div class="machine-icon">
                    ${ReactDOMServer.renderToString(icon)}
                </div>
                <div class="machine-data">
                    <span class="machine-code typography">${machine.code}</span>
                    <span class="machine-name typography">${machine.name}</span>
                </div>
            </div>
        `;
        },
    },
},
```

Figura 46: Configuración del eje vertical (eje de coordenadas) con nombres únicos.

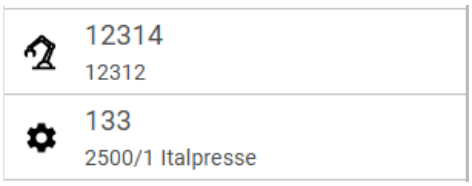

Figura 47: Renderizado personalizado de las máquinas en el eje Y.

Series: Tenemos dos series: la primera, contiene las órdenes de fabricación, las cuales se pueden arrastrar horizontalmente (*draggableX*), pero no modificar su duración y la segunda, representa los bloques/slots donde la máquina no está disponible (ver Figura 48).

```
series: [
    {
        data: preparedData,
        dragDrop: {
            draggableEnd: false,
            draggableStart: false,
            draggableX: true,
        },
        point: {
            events: {
                drop: handleDrop,
            },
        },
        pointWidth: 50,
    },
    {
        data: nonProductiveSlots,
        dataLabels: {
            useHTML: true,
            align: 'center',
            allowOverlap: true,
        },
        pointWidth: 54,
    },
],
```

Figura 48: Opciones para las series del gráfico (órdenes y slots no productivos).

Representación personalizada de las órdenes y los periodos no productivos.

Uno de los principales retos del desarrollo de la interfaz del secuenciador fue la representación visual de las órdenes de fabricación y los periodos de no productividad dentro del gráfico Gantt. Highcharts no permite modificar directamente el contenido HTML de los puntos (*points*) ni incrustar componentes React dentro de ellos, lo cual limitaba las posibilidades de personalización y adaptación al diseño deseado.

Para solventar esta limitación, se utilizó *ReactDOM.createPortal*, una herramienta que permite renderizar componentes de React en nodos DOM fuera de la jerarquía principal del componente. De esta forma, fue posible insertar componentes personalizados dentro de los elementos *div* generados dinámicamente por Highcharts.

Implementación de la solución:

1. Generación de contenedores HTML vacíos para los puntos: En la configuración del gráfico (*plotOptions.series.dataLabels.formatter*), se generó un *div* con un id único asociado a cada punto del gráfico como muestra la Figura 49.

```
formatter: function() {
    return `<div id="${this.point.id}" style="width: ${this.point.shapeArgs?.width}px;
            height: ${this.point.shapeArgs?.height}px;"></div>`;
},
```

Figura 49: Generación de contenedores HTML para etiquetas personalizadas.

2. Detección de los puntos renderizados: Se añadió un manejador al evento render del gráfico, el cual se dispara cada vez que el gráfico es renderizado. En este manejador se recorre la lista de puntos visibles del gráfico y se genera para cada uno un componente *CustomLabelGantt* (ver código en Figura 50).

```
events: {
    render() {
        const points = this.series[0].points.map((point, i) => <CustomLabelGantt key={`point-${i}`} data={point} />);
        setCustomPoints(points);
    },
}
```

Figura 50: Manejador render para insertar etiquetas personalizadas.

3. Uso de *createPortal* para insertar el contenido React dentro del gráfico: El componente *CustomLabelGantt* se encarga de buscar el *div* correspondiente a cada punto (mediante *document.getElementById*) y renderizar ahí, usando *createPortal*, el componente *ItemTemplate*, que representa visualmente cada orden (ver Figura 51).

```
useEffect(() => {
    const checkElement = () => {
        const element = document.getElementById(data.id?.toString() ?? '');
        if (element) setElement(element);
    };
    checkElement();
    const interval = setInterval(checkElement, 100);
    return () => clearInterval(interval);
}, [data.id]);

if (!element) return null;

return createPortal(
    <ItemTemplate {...props} />,
    element
);
```

Figura 51: Renderizado de órdenes (*ItemTemplate*) mediante *createPortal*.

4. Renderizado final dentro del componente principal: Finalmente, la lista *customPoints* se renderiza directamente dentro del JSX del componente, asegurando que los elementos React aparecen sincronizados con los puntos del gráfico como ilustra la Figura 52.

```
return (
    <>
        {customPoints}
        <div
            className="highcharts-light"
            style={{
                height: 'calc(100% - 50px)',
                overflowX: 'hidden',
                overflowY: 'scroll',
                marginTop: '5px',
                marginLeft: '5px',
            }}
        >
            <HighchartsReact highcharts={Highcharts} constructorType={'ganttChart'} options={options} ref={chartRef} />
        </div>
        {openMove && <MoveModal open={openMove} setOpen={setOpenMove} moveInfo={moveInfo} updateDates={updateDates} />}
    </>
);
```

Figura 52: Renderizado de las órdenes *custom* en la interfaz.

En contraste con las órdenes, los slots de no productividad no requerían un componente personalizado, sino simplemente una representación visual diferenciada y clara. Para ello, se optó por una solución más simple, basada exclusivamente en estilos CSS personalizando directamente el *div* creado con *createPortal*.

En el *formatter* de las etiquetas para la serie correspondiente a los slots no productivos, se generó un *div* con una clase específica *non-productive-label* (ver Figura 53)

```
formatter: function() {
    return `<div class="non-productive-label" style="width: ${this.point.shapeArgs?.width}px;
    height: ${this.point.shapeArgs?.height}px;"> </div>`;
},
```

Figura 53: Generación de contenedores HTML para los *slots* no productivos.

La clase *non-productive-label* aplica un patrón de rayado diagonal mediante *background-image*, así como bordes laterales (ver código en Figura 54) para representarlos como se hacía en la interfaz original.

```css
.non-productive-label {
    background-image: linear-gradient(45deg, #d3d3d3 25%, transparent 25%, transparent 50%, #d3d3d3 50%, #d3d3d3 75%,
      transparent 75%, transparent);
    background-size: 20px 20px;
    border-right: 1px solid #000 !important;
    border-left: 1px solid #000 !important;
    margin-left: 7px;
}
```

Figura 54: Estilo visual de los *slots* no productivos.

Mejora de la visibilidad del secuenciador (ocultar el panel de búsqueda): Una vez terminado el secuenciador, mientras probaba su funcionamiento, se me ocurrió una nueva funcionalidad y se lo propuse a mi tutor en la empresa. La idea consistía en permitir al usuario ocultar el panel lateral izquierdo de búsqueda de órdenes, de modo que el área del diagrama Gantt pudiera ocupar más espacio horizontal en pantalla, facilitando así la visualización de tareas y máquinas, especialmente en entornos con resoluciones limitadas.

La propuesta fue bien recibida por el equipo, por lo que procedí a su implementación. Para ello, incorporé un botón que permite alternar dinámicamente la visibilidad del panel, utilizando un estado local. En el *grid* principal del componente, se renderiza condicionalmente el panel de búsqueda (*SearchItems*) si *showSearch* es true. En caso contrario, se muestra un botón flotante para volver a mostrar el panel (ver código en Figura 55).

```jsx
const Sequencer = () => {
  const [showSearch, setShowSearch] = useState(true);
  return (
      <>
          {(isSequencerDataLoading || isItemsToSequenceLoading) && <ActionProgressReport />}
          <Grid container spacing={2} style={{ margin: 0, width: '100%', height: '100%' }}>
              {showSearch && <SearchItemns onToggle={() => setShowSearch(false)} />}
              {!showSearch && (
                  <IconButton
                      onClick={() => setShowSearch(true)}
                      style={{
                          left: 0,
                          color: '#FFF',
                          width: '22px',
                          height: '50px',
                          zIndex: 1000,
                          position: 'absolute',
                          top: '50%',
                          marginLeft: '16px',
                          borderRadius: '4px',
                          transform: 'translate(-50%, -50%)',
                          backgroundColor: '#48D8B2',
                      }}
                      size="small"
                  >
                      <ChevronRightIcon />
                  </IconButton>
              )}
              ...
              <FilterOptions />
              <GanttChart />
              <SequencerModal />
          </Grid>
      </>
  );
}
```

Figura 55: Botón flotante para ocultar el panel lateral de búsqueda.

La Figura 56 muestra la interfaz del secuenciador con el panel de búsqueda de órdenes oculto. Además, se puede observar cómo el nuevo botón presenta una flecha hacia la derecha, indicando al usuario que puede volver a mostrar el panel pulsando sobre él.

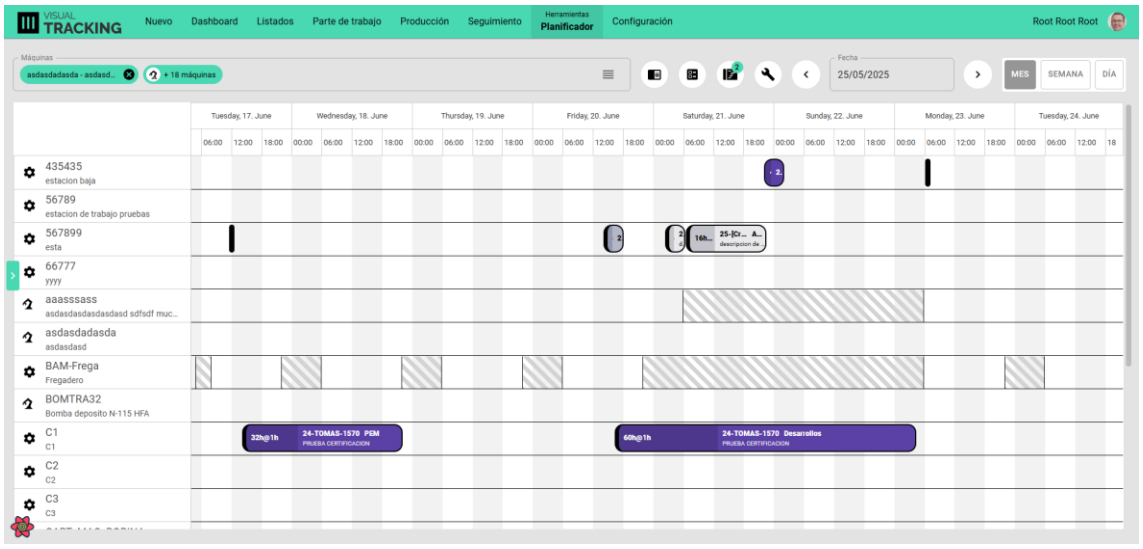

Figura 56: Interfaz del secuenciador con el panel de búsqueda de órdenes oculto.

Finalmente, la Figura 57 muestra el resultado final de la interfaz del secuenciador desarrollada con Highcharts, en la que se presenta el panel de búsqueda de órdenes visible en el lateral izquierdo.

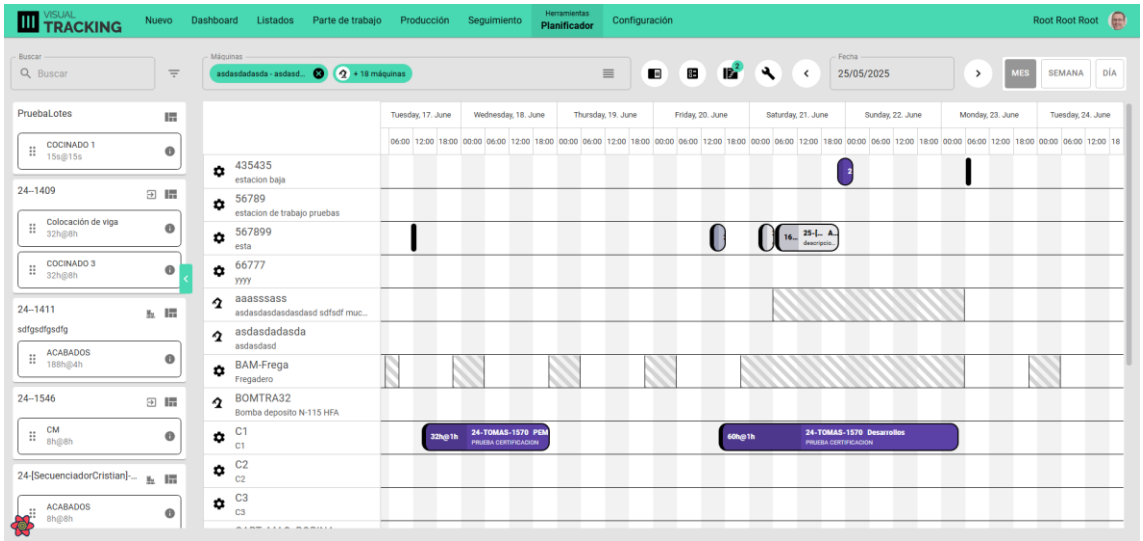

Figura 57: Resultado final de la interfaz del secuenciador.

5. VALIDACIÓN Y RESULTADOS

En este apartado se presentan los procedimientos y resultados obtenidos durante la fase de validación del proyecto, con el objetivo de comprobar el correcto funcionamiento de las mejoras implementadas en las interfaces de usuario.

5.1 Interfaz Kanban de seguimiento de órdenes

A continuación, se detallan los objetivos, la metodología empleada y los principales resultados durante la validación de la interfaz Kanban.

5.1.1 Objetivo de la validación

Dado que las modificaciones realizadas fueron exclusivamente a nivel de diseño y organización visual, el objetivo de la validación de la nueva interfaz Kanban fue garantizar que las funcionalidades principales se mantuvieran operativas, asegurando que la experiencia de usuario mejorara sin comprometer la funcionalidad del sistema.

Al tratarse de un rediseño centrado en la mejora estética, la legibilidad y la organización de la información, la validación buscó comprobar que:

- Los elementos visuales (colores, jerarquía, espaciado) se aplican correctamente.
- Las interacciones principales (crear, editar, filtrar órdenes) siguen funcionando de forma fluida y sin errores.
- Las nuevas funcionalidades añadidas (como la codificación por colores según prioridad o el orden automático de tarjetas) funcionan como se espera.

5.1.2 Metodología

Para evaluar el comportamiento de la nueva interfaz y detectar posibles errores en su funcionamiento, se optó por utilizar Cypress, una herramienta de *testing end-to-end* que permite simular interacciones reales del usuario sobre una aplicación web. Gracias a esta metodología, se pudo automatizar la validación de algunos casos de uso, asegurando que las funcionalidades clave respondieran de forma correcta ante distintas acciones del usuario.

Las pruebas se ejecutaron sobre un entorno local de desarrollo. Además de las pruebas automatizadas, se realizó una validación visual manual para verificar la correcta integración de estilos, el uso del color y la legibilidad general de los elementos.

5.1.3 Pruebas realizadas

Se diseñaron tres pruebas principales:

- Creación de una nueva orden con baja prioridad, verificación de su aparición en la interfaz y comprobación del color de borde correspondiente.
- Edición de la orden existente, modificando la prioridad a alta y validando el cambio visual (color rojo en el borde).
- Filtrado por descripción, asegurando que la búsqueda devuelve correctamente la orden esperada con su información visual intacta.

Además, se comprobó que los contadores de tarjetas por fase se actualizaban dinámicamente y que el comportamiento de ordenación por prioridad funcionaba correctamente. La codificación de estas pruebas *end to end* con Cypress se pueden ver en el Anexo A: Codificación de test end to end con Cypress.

5.1.4 Resultados

La validación permitió confirmar que la nueva interfaz Kanban mantiene la funcionalidad original sin ningún problema, a la vez que mejora notablemente la claridad visual, la organización de la información y la experiencia de usuario. Se logró una presentación más moderna, coherente y alineada con los principios de diseño establecidos en el capítulo 2.

Además, todos los requisitos funcionales y no funcionales definidos en el apartado 4.1.2 fueron satisfechos. El sistema permite una interacción fluida, rápida y más comprensible para los usuarios industriales, cumpliendo así los objetivos de modernización y mejora planteados al inicio del proyecto.

5.2 Secuenciador

A continuación, se describen los objetivos planteados, el enfoque metodológico y los resultados obtenidos en la validación de la interfaz del secuenciador de órdenes.

5.2.1 Objetivo de la validación

El objetivo principal de la validación del secuenciador de órdenes fue evaluar el rendimiento de la nueva implementación desarrollada con la librería Highcharts Gantt, en comparación con la versión anterior. La validación se centró especialmente en comprobar la fluidez de la navegación, la capacidad de respuesta del sistema y la escalabilidad visual al representar un número creciente de máquinas y tareas en el eje temporal.

La motivación de este rediseño fue solventar los problemas de rendimiento detectados en la versión previa del secuenciador, en la cual el desplazamiento

horizontal provocaba recargas constantes y una experiencia de usuario poco fluida. De esta forma, el objetivo de la validación fue determinar si la nueva solución ofrece una mejora real en términos de estabilidad, eficiencia gráfica y comportamiento interactivo, especialmente en escenarios de alta carga.

5.2.2 Metodología

A diferencia de la interfaz Kanban, en esta interfaz no se aplicaron pruebas automatizadas, ya que muchas de las interacciones clave del secuenciador (*scroll* horizontal, zoom dinámico, arrastre de tareas) no son fácilmente emulables mediante herramientas como Cypress. En este caso, se llevó a cabo una validación manual, utilizando un entorno de desarrollo local con datos simulados.

El enfoque consistió en probar de forma progresiva la interfaz con un número creciente de máquinas y tareas, observando el comportamiento del sistema en términos de fluidez, estabilidad y capacidad de respuesta.

Las pruebas se realizaron desde un navegador Google Chrome sobre un portátil con prestaciones estándar, con el objetivo de aproximarse a un entorno realista de uso por parte de operarios o planificadores.

5.2.3 Pruebas realizadas

A continuación, se detallan los principales escenarios evaluados:

- Fluidez del *scroll* horizontal y zoom con la rueda del ratón.
- Visualización y claridad de las tareas: se evaluó la representación de las órdenes y los bloques de no productividad, incluyendo su posición en el tiempo, su agrupación por máquina y la correcta aplicación de estilos visuales.
- Interacción mediante arrastrar y soltar: se probó el mecanismo de reubicación de tareas a lo largo del eje temporal.
- Gestión de solapamientos: se forzó intencionalmente la superposición de dos tareas sobre una misma máquina y franja horaria, validando que el sistema detectaba el conflicto y mostraba el modal de aviso para resolverlo.
- Carga progresiva de datos: se probaron diferentes cargas, aumentando progresivamente el número de máquinas representadas (20, 50, 70, 100, 150).

5.2.4 Resultados

Los resultados de la validación mostraron un comportamiento correcto a nivel funcional: las tareas se representan con claridad, los bloques de no productividad se dibujan correctamente en el eje temporal, el mecanismo de arrastre funciona con precisión y el sistema detecta adecuadamente los conflictos entre tareas, mostrando el modal correspondiente. También se confirmó que el zoom personalizado y el

desplazamiento horizontal permiten navegar por el diagrama de forma intuitiva, al menos hasta cierto nivel de carga.

Sin embargo, el rendimiento del secuenciador no experimentó la mejora esperada. Las pruebas revelaron que la interfaz se mantiene fluida y estable hasta aproximadamente 70 máquinas representadas simultáneamente. A partir de ese punto, la fluidez comienza a empeorar de forma progresiva: el *scroll* horizontal se vuelve menos ágil, el zoom responde con cierto retardo y el redibujado de tareas se percibe más lento. En escenarios con más de 100 máquinas, estas deficiencias se vuelven más notables, afectando potencialmente a la experiencia de uso en entornos reales de alta carga.

Estos problemas de rendimiento tienen su explicación en cómo está construida la solución actual. Highcharts Gantt utiliza tecnología SVG para renderizar los elementos del diagrama, lo que significa que cada tarea, franja de inactividad, etiqueta o línea se traduce en un nodo SVG dentro del DOM del navegador. Este enfoque funciona bien con cantidades moderadas de datos, pero cuando el número de máquinas o tareas aumenta significativamente, el navegador debe gestionar cientos o miles de nodos SVG simultáneamente, lo que impacta directamente en la fluidez de la interfaz.

Además, Highcharts no incorpora técnicas de virtualización, por lo que todos los elementos del gráfico se renderizan, estén visibles o no, lo cual incrementa la carga gráfica incluso cuando el usuario solo ve una pequeña porción del diagrama.

A esto se suma el hecho de que, en esta implementación, cada tarea del diagrama no solo está representada por el punto (*point*) estándar de la librería, sino que también se superpone un componente React personalizado mediante *createPortal* para mostrar información adicional. Esto implica que por cada tarea se están generando dos representaciones distintas, lo que multiplica el número de elementos activos en el DOM y añade una capa extra de complejidad en el renderizado.

Como resultado tenemos que, aunque la nueva implementación del secuenciador ha permitido conservar la estética original y mejorar aspectos de usabilidad y organización, el sistema no escala adecuadamente a partir de cierto volumen de datos, lo cual representa una limitación importante. Cabe destacar que esta limitación ya estaba presente en la versión original; sin embargo, el uso de Highcharts hace que el rendimiento comience a degradarse ligeramente antes, lo que constituye un aspecto a tener en cuenta para futuras optimizaciones.

6. CONCLUSIONES Y LÍNEAS DE TRABAJO FUTURO

Este trabajo se ha desarrollado en un entorno profesional real, con el objetivo de mejorar la interfaz de usuario y la experiencia de usuario de varias interfaces del producto EMI Suite 4.0. A lo largo del proyecto se han analizado las limitaciones visuales y funcionales de las interfaces existentes, y se han planteado soluciones concretas fundamentadas en principios de diseño centrado en el usuario, guías reconocidas y en teorías psicológicas y heurísticas ampliamente aceptadas en el ámbito del diseño UI/UX.

Los objetivos planteados al inicio del proyecto se han cumplido satisfactoriamente, y en algunos casos incluso se han superado, incorporando mejoras que han sido bien acogidas por el equipo de Soincon y que se han integrado en el entorno real de la aplicación.

Además de cumplir los objetivos técnicos, el proyecto representa una aportación personal significativa al haber trasladado criterios avanzados de diseño UI/UX a un entorno industrial real, con restricciones concretas. Se ha aplicado un enfoque centrado en el usuario sin contacto directo con usuarios finales, lo cual exigió un esfuerzo adicional de empatía, análisis contextual y adaptación metodológica.

Cabe destacar que este TFG no solo cumple un objetivo práctico, sino que también puede servir como una guía de referencia para futuros desarrollos centrados en UI/UX, especialmente en contextos industriales. En el Capítulo 2 se recogen fundamentos teóricos y principios de diseño que han guiado las decisiones tomadas a lo largo del proyecto, y que pueden ser reutilizados o ampliados en trabajos similares.

En cuanto al trabajo realizado, a pesar de los avances logrados, el proyecto deja abiertas varias líneas de trabajo que podrían desarrollarse en el futuro para seguir mejorando la experiencia de usuario y la robustez del sistema:

- **Validación con usuarios reales en planta**: Aunque la mejora de la interfaz se ha realizado siguiendo principios reconocidos de diseño y heurísticas de usabilidad, no ha tenido la oportunidad de llevar a cabo una validación con usuarios finales reales. Para simular este proceso, se ha diseñado un artefacto de pruebas, que incluye un procedimiento de evaluación y un cuestionario estructurado que puede ser utilizado para realizar pruebas de usabilidad en entornos reales. Este artefacto se encuentra documentado en el Anexo B: Artefacto de pruebas con usuarios reales.
- **Evaluación de accesibilidad**: Otra futura línea de trabajo consiste en aplicar herramientas como Lighthouse o axe-core para evaluar y asegurar que la interfaz cumple con los criterios de accesibilidad definidos por los estándares WCAG 2.1, garantizando un acceso equitativo a todos los usuarios.

7. BIBLIOGRAFÍA

Aliyev, A. THE APPLICATION OF GESTALT THEORY IN UX/UI DESIGN: PRINCIPLES AND THEIR IMPORTANCE FOR USERS. DOI:10.51582/interconf.19-20.12.2024

Apple Inc. (s. f.). Human Interface Guidelines. Apple Developer. https://developer.apple.com/design/human-interface-guidelines/

Bouza, M., Crivaro, E. (2022). *Principios de Diseño UX y diseño UI.* Recuperado el 22 de abril de 2025, de https://medium.com/@nerdas.estudio/principios-de-dise%C3%B1o-ux-ui-18f04ec0e9ae

Cha, A. P., & Romli, A. (2010). Human-computer interaction of design rules and usability elements in expert system for personality-based stress management. *International Journal of Intelligent Computing Research (IJICR)*, *1*(1/2), 33-42.

Fernández Casado, P. E. (2021). *UX design: Hazlo fácil pensando en el usuario*. Ra-Ma.

Garrett, J. J. (2010). *The Elements of User Experience: User-Centered Design for the Web and Beyond*. New Riders.

Google. (2014). Material Design. https://m2.material.io/

Hamidli, N. (2023). Introduction to UI/UX design: key concepts andrinciples. *Preuzeto*, *28*, 2024.

Hassenzahl, M., & Tractinsky, N. (2006). User experience-a research agenda. *Behaviour & information technology*, *25*(2), 91-97.

International Organization for Standardization. (2018). ISO 9241-11:2018: Ergonomics of human-system interaction – Part 11: Usability: Definitions and concepts. https://www.iso.org/standard/63500.html

International Organization for Standardization. (2019). ISO 9241-210:2019: Ergonomics of human-system interaction – Part 210: Human-centred design for interactive systems. https://www.iso.org/standard/77520.html

International Society of Automation. (2010). *Enterprise-control system integration – Part 1: Models and terminology (ANSI/ISA-95.00.01-2010)*. ISA.

Krug, S. (2015). *No me hagas pensar: Una aproximación a la usabilidad en la web* (6ª ed.). Prentice Hall.

Mäkelä, A., & Fulton Suri, J. (2001, June). Supporting users' creativity: Design to induce pleasurable experiences. In *Proceedings of the International Conference on Affective Human Factors Design*

Nielsen, J. (1994, abril 24). 10 usability heuristics for user interface design. Nielsen Norman Group. https://www.nngroup.com/articles/ten-usability-heuristics/

Nielsen J. (2024). Visual UI Design with Gestalt Principles. UX Tigers Blog. https://www.uxtigers.com/post/gestalt-principles

Organización Mundial de la Salud. (2023, 7 de marzo). Discapacidad y salud. Organización Mundial de la Salud. https://www.who.int/es/news-room/fact-sheets/detail/disability-and-health

Organización Mundial de la Salud. (2023, 10 de agosto). Ceguera y discapacidad visual. Organización Mundial de la Salud. https://www.who.int/es/news-room/fact-sheets/detail/blindness-and-visual-impairment

Organización Mundial de la Salud. (2025, 26 de febrero). Sordera y pérdida de la audición. Organización Mundial de la Salud. https://www.who.int/es/news-room/fact-sheets/detail/deafness-and-hearing-loss

Puente, O. (2023). *Los 8 principios del diseño UI: Introducción al diseño UI.* Recuperado el 22 de abril de 2025, de https://oliverpuente.com/ui-ux/los-8-principios-del-disenoui/?srsltid=AfmBOoqLAsS58iHVqXQ3BOocZhwPtKXsZ20i3CW9VUlO3ZMT0vJjUivd

Shneiderman, B. (1998). Designing the user interface: Strategies for effective human-computer interaction. Addison-Wesley.

Torres, D. (2024). *5 principios básicos del UX Design*. Recuperado el 22 de abril de 2025, de https://platzi.com/blog/principios-basicos-ux-design/

Viáfara Gálvez, L. (2021). *Industria 4.0, gestión del conocimiento*. Editorial Universidad Icesi.

Villani, V., Sabattini, L., Czerniak, J. N., Mertens, A., Vogel-Heuser, B., & Fantuzzi, C. (2017). Towards modern inclusive factories: A methodology for the development of smart adaptive human-machine interfaces. https://doi.org/10.48550/arxiv.1706.08467

8. ANEXOS

Anexo A: Codificación de test end to end con Cypress

```
describe('Test Kanban Interface', () => {
    before(() => {
        cy.login();
    });

    it('frontpage loads', () => {
        cy.visit('http://localhost:3001/visual-tracking/dashboard');
    });

    it('añade una orden con prioridad baja, verifica que se añade y verifica el color del borde', () => {
        let initialCount;
        cy.get('.list-title-button') // Obtener el contador inicial de ordenes para la lista de ofertas
            .contains('Oferta')
            .parent()
            .find('span')
            .eq(1) // el segundo <span> es el contador de ordenes
            .invoke('text')
            .then(text => {
                initialCount = parseInt(text);
            });

        cy.contains('Nuevo').click(); // Clic en el botón "Nuevo" para crear una nueva orden

        cy.get('input[name="priorityId"]') // Sleccionar baja prioridad
            .parent()
            .click();
        cy.get('ul[role="listbox"]')
            .contains('Baja')
            .click();

        cy.get('[name="personId"]') // Seleccionar Prueba1 como cliente
            .parent()
            .click();
        cy.get('ul[role="listbox"]')
            .contains('Prueba1')
            .click();

        cy.wait(1000);

        cy.get('input[name="code"]') // Introducir el codigo de la orden
            .clear()
            .type('Prueba Cypress');

        cy.get('textarea[name="description"]') // Introducir la descripción de la orden
            .clear()
            .type('Descripción de prueba para Cypress');

        cy.wait(500);

        cy.contains('Guardar').click(); // Clic en "Guardar" para crear la orden

        cy.wait(500);

        cy.contains('Prueba Cypress').should('exist'); // Verificar que la orden aparece con el codigo y cliente indicados
        cy.contains('Prueba Cypress')
            .parents('.item-card')
            .should('contain', 'Prueba1');

        cy.contains('Prueba Cypress') // Verificar que el borde izquierdo de la tarjeta es verde (prioridad baja)
            .parents('.item-card')
            .should('have.css', 'border-left-color', 'rgb(72, 216, 178)');

        cy.get('.list-title-button') // Verificar que el contador de la lista "Oferta" ha aumentado en 1
            .contains('Oferta')
            .parent()
            .find('span')
            .eq(1)
            .invoke('text')
            .then(updatedText => {
                const updatedCount = parseInt(updatedText);
                expect(updatedCount).to.eq(initialCount + 1);
            });
    });

    it('edita la orden y cambia su prioridad a alta, comprobando el borde rojo', () => {
        cy.contains('Prueba Cypress').click(); // Abrir la orden haciendo clic sobre ella

        cy.get('input[name="priorityId"]') // Cambiar la prioridad a "Alta"
            .parent()
            .click();
        cy.get('ul[role="listbox"]')
            .contains('Alta')
            .click();

        cy.contains('Guardar').click(); // Guardar los cambios
        cy.wait(500);
        cy.contains('Prueba Cypress') // Verificar que el borde izquierdo de la tarjeta es rojo (prioridad alta)
            .parents('.item-card')
            .should('have.css', 'border-left-color', 'rgb(217, 83, 79)');
    });

    it('filtra por descripción y verifica que la orden editada aparece con prioridad alta', () => {
        cy.get('button[title="Filtro"]') // Abrir los filtros
            .parent()
            .click();

        cy.get('input[placeholder="Descripción"]') // Introducir la descripción de la orden para filtrar
            .click()
            .type('Descripción de prueba para Cypress');
        cy.wait(500);

        cy.contains('Guardar').click(); // Aplicar el filtro

        cy.contains('Prueba Cypress').should('exist'); // Verificar que contiene el cliente y el codigo correctos
        cy.contains('Prueba Cypress')
            .parents('.item-card')
            .should('contain', 'Prueba1');

        cy.contains('Prueba Cypress') // Verificamos que el borde sigue siendo rojo (alta prioridad)
            .parents('.item-card')
            .should('have.css', 'border-left-color', 'rgb(217, 83, 79)');
    });
});
```

Anexo B: Artefacto de pruebas con usuarios reales

Este artefacto tiene como finalidad proporcionar un protocolo estandarizado para llevar a cabo una evaluación de usabilidad de la interfaz Kanban rediseñada en EMI Suite 4.0 con usuarios reales. El objetivo es obtener datos cualitativos y cuantitativos sobre la experiencia de uso, detectar posibles puntos de mejora y validar las decisiones de diseño tomadas durante el desarrollo del proyecto.

B.1 Procedimiento de evaluación

A continuación, se describe el procedimiento recomendado para realizar las pruebas con usuarios:

1. Selección de participantes
    - Perfil ideal: operarios, técnicos o responsables de producción que utilicen habitualmente EMI Suite 4.0.
    - Muestra recomendada: entre 5 y 10 usuarios.
2. Preparación del entorno
    - Se utilizará una instancia de prueba de EMI Suite 4.0 con datos ficticios.
    - Se habilitará solo el módulo de Visual Tracking.
    - El evaluador deberá tener disponible este cuestionario impreso o digital.
3. Escenarios de prueba

    Interfaz Kanban:

    - Localizar una orden de fabricación concreta usando los filtros.
    - Crear una nueva orden de fabricación.
    - Editar una orden existente.
    - Ordenar las tarjetas por prioridad utilizando el botón correspondiente.
    - Interpretar correctamente el color del borde para identificar órdenes urgentes.

    Secuenciador:

    - Navegar horizontalmente por el diagrama de Gantt utilizando scroll y zoom.
    - Identificar las órdenes de fabricación y su información.
    - Arrastrar una orden a otro punto del eje temporal y confirmar el cambio.
    - Forzar un solapamiento y responder al modal de conflicto.

- Ocultar o mostrar el panel lateral de búsqueda.

4. Observación

   Durante la ejecución de las tareas se deberá observar:

   - Tiempo estimado para completar cada tarea.
   - Dificultades detectadas.
   - Comentarios de los usuarios.
   - Fluidez y velocidad de interacción.

Una vez finalizadas las tareas, cada usuario completará un cuestionario de evaluación como el que se muestra a continuación.

B.2 Cuestionario de evaluación

Por favor, marque su grado de acuerdo con las siguientes afirmaciones (1 = Totalmente en desacuerdo, 5 = Totalmente de acuerdo):

Num.	Afirmación	1	2	3	4	5
1	Las interfaces son visualmente claras y comprensibles.					
2	Me ha resultado fácil encontrar la información o funcionalidad que buscaba.					
3	El uso de colores, iconos y organización visual facilita la interpretación de la información.					
4	Las interacciones con la interfaz son intuitivas y fluidas.					
5	La experiencia de uso ha sido satisfactoria y agradable.					
6	Utilizaría estas interfaces en mi trabajo diario sin dificultad.					

Preguntas abiertas:

- ¿Qué aspectos te han parecido más útiles o cómodos de las nuevas interfaces?

  [respuesta]

- ¿Qué mejorarías o cambiarías en alguna de las dos interfaces?

  [respuesta]

- ¿Hubo algo que te confundiera o te resultara poco claro durante el uso?

  [respuesta]