Universidad del Valle de Guatemala
Facultad de Ingeniería

Departamento de Ciencia de la Computación
Ingeniería de Software I

DESIGN BETTER

PLATAFORMA DE DISEÑO Y PERSONALIZACIÓN DE MODA

Pablo José Méndez Alvarado – 23975
Luis Fernando Palacios López – 239333
Roberto Samuel Nájera Marroquín – 23781
André Emilio Pivaral López – 23574

Catedrático: Erick Francisco Marroquín Rodríguez

Sección: 20

Nueva Guatemala de la Asunción, 19 de marzo de 2025

Índice

Resumen

El presente trabajo aborda el desafío que enfrenta una diseñadora de moda independiente al crear prendas personalizadas para distintos tipos de cuerpo, asegurando un equilibrio entre individualidad, calidad y eficiencia en el proceso de diseño. La creciente demanda de moda personalizada y la digitalización del diseño hacen que sea necesario un sistema que facilite la personalización, visualización y comunicación con los clientes.

El objetivo principal de este proyecto es desarrollar una solución que permita modelar, simular y personalizar prendas de manera eficiente, optimizando el flujo de trabajo y mejorando la experiencia tanto de la diseñadora como del cliente. Para ello, se busca implementar herramientas que agilicen la toma de medidas, proporcionen sugerencias de estilo y ofrezcan visualizaciones realistas.

Además, el proyecto apunta a reducir costos y tiempos en la creación de prototipos, mejorar la satisfacción del cliente y diferenciarse en un mercado competitivo. La solución deberá ser intuitiva y accesible, permitiendo a la diseñadora interactuar con los clientes de manera ágil, recopilar retroalimentación y generar representaciones fieles de las prendas antes de su confección.

Introducción

Descripción de la Persona

El proyecto está dirigido a una diseñadora de moda independiente que enfrenta desafíos al crear prendas personalizadas para distintos tipos de cuerpo. Su trabajo requiere precisión en la toma de medidas, visualización realista de los diseños y una comunicación fluida con los clientes para garantizar su satisfacción.

Dado que trabaja de manera autónoma o con un equipo reducido, la diseñadora necesita herramientas que le permitan agilizar su proceso de diseño, optimizar costos en prototipos y brindar experiencias de personalización más atractivas y eficientes para sus clientes.

Descripción de la Idea

La idea central del proyecto es desarrollar una solución tecnológica que facilite el diseño y personalización de prendas mediante herramientas de modelado y simulación digital.

Actualmente, la diseñadora enfrenta dificultades para visualizar sus creaciones en diferentes tipos de cuerpo, lo que puede generar incertidumbre en los clientes y múltiples iteraciones en el diseño.

La ausencia de herramientas accesibles y realistas limita la capacidad de personalización y la eficiencia en la toma de decisiones. Implementar un sistema que permita visualizar de manera precisa las prendas antes de su confección ayudará a optimizar tiempos, reducir costos y mejorar la experiencia tanto para la diseñadora como para sus clientes.

Objetivo General

Desarrollar una solución tecnológica que permita a la diseñadora de moda modelar, simular y personalizar prendas de manera eficiente, optimizando su proceso de diseño y mejorando la experiencia del cliente.

Objetivos Específicos

• Facilitar los procesos de toma de medidas y personalización de prendas para diferentes tipos de cuerpo.

• Implementar visualizaciones realistas en 3D para ayudar a los clientes a previsualizar las prendas antes de la confección.

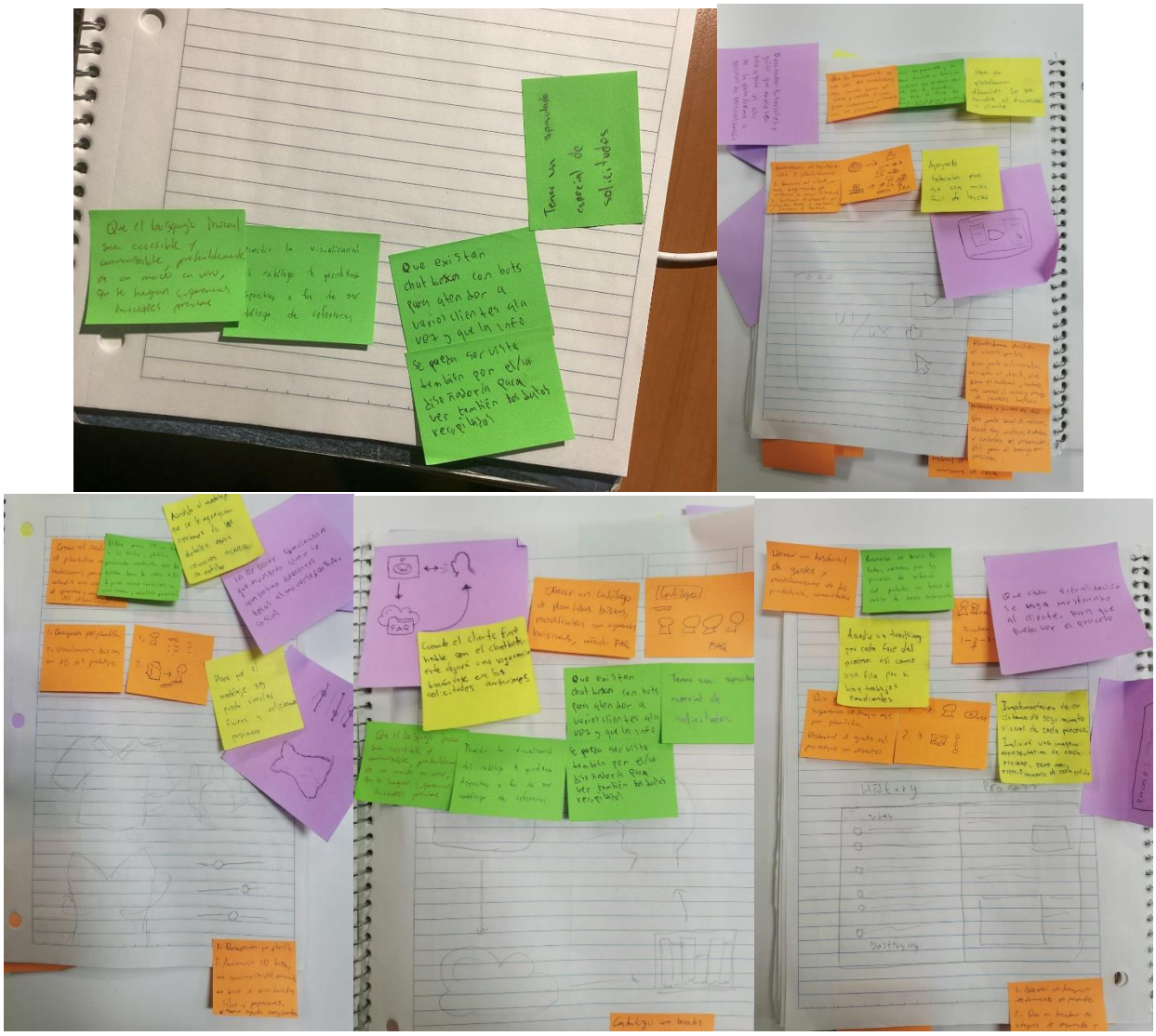
• Agilizar el proceso de iteración en los diseños, reduciendo la cantidad de muestras físicas necesarias.

• Mejorar la comunicación entre la diseñadora y sus clientes, a través de herramientas de software interactivas.

• Reducir costos y tiempos en la creación de prototipos, optimizando el flujo de trabajo de la diseñadora.

Design Studio

A continuación, se presentan las evidencias de la implementación de la metodología Design Studio en nuestro proyecto.



Realizamos algunas reuniones entre los integrantes del grupo para realizar la técnica, escribiendo nuestras ideas en pósits para luego discutirlas y llegar propuestas finales para solucionar las oportunidades previamente analizadas.

Consideramos que fue un proceso efectivo donde logramos combinar las ideas más útiles en una solución convincente, como se aprecia en las imágenes adjuntas. Además, planteamos nuestras historias de usuario en base a las necesidades identificadas para el uso y funcionalidad requeridas de nuestros diferentes usuarios.

Prototipos

Para agilizar el proceso se decidió unificar los prototipos de todas las ideas en una sola presentación para cada iteración. El futuro usuario entrevistado fue la diseñadora Gabriela Nájera. No me dio permiso de tomar fotografía o video, pero si me dio permiso de grabar sonido. Los audios de las pruebas se encuentran en el repositorio de GitHub, en la carpeta de audio correspondiente al Corte II.

Primera Versión

Esta primera versión del prototipo está basada en las ideas finales del Design Studio:

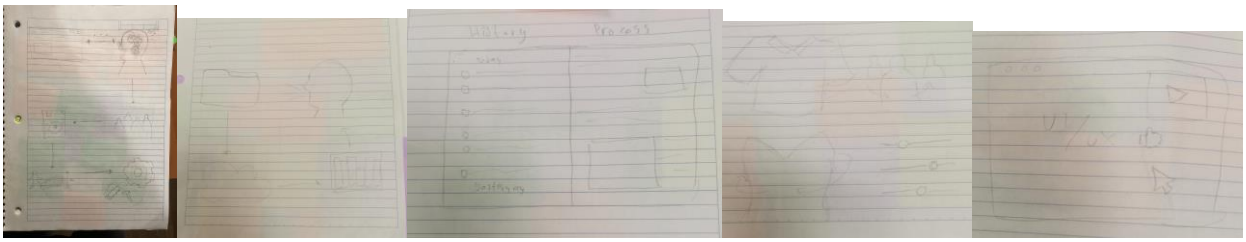
1. Biblioteca de medidas y plantillas: Un sistema que recomienda diseños en base al tipo de cuerpo del usuario.

2. Catálogo con bocetos y FAQ: Para mejorar la comunicación entre diseñador y cliente.

3. Sistema de seguimiento de pedidos: Un tracker visual con estimaciones de tiempo y actualizaciones del diseño.

4. Simulación 3D de prendas: Modelos en 3D con opciones de visualización de diferentes telas.

5. Plataforma con secciones diferenciadas: Una parte para interacción remota y otra para el diseñador con herramientas de control.



Retroalimentación

Hay un interés en biblioteca de medidas y plantillas con modelos estándar y medidas personalizadas que ayuda con la visualización del resultado. Además, conviene añadir una ayuda personalizada como un *chatbot*.

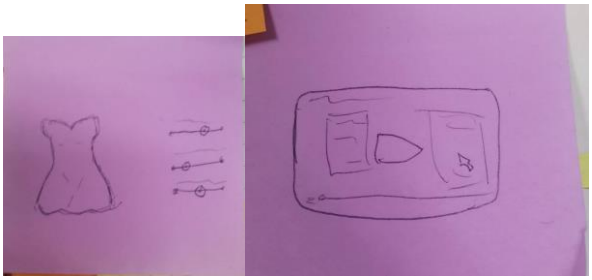
Segunda Versión

Esta segunda versión del prototipo está basada en la retroalimentación recibida:

Cambios Realizados

• Modelos estándar predefinidos y posibilidad de ingresar medidas personalizadas para una recomendación más precisa.

• Integración de un asistente virtual o *chatbot* para facilitar la navegación y recomendaciones.



Retroalimentación

Se nota que el hecho de poder ingresar medidas personalizadas mejora la precisión del resultado obtenido, además que la integración de un *chatbot* con respuestas elaboradas puede facilitar el uso de esta herramienta.

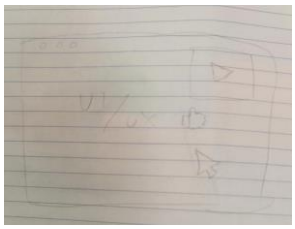
Tercera Versión

Esta versión del prototipo está basada en la continuación de la retroalimentación:

Cambios Realizados

• La biblioteca de plantillas es basada en moldes básicos con opciones de personalización según volumen y proporciones del usuario.

• El *chatbot* inteligente viene con respuestas guiadas y posibilidad de derivar a un asesor humano cuando sea necesario.



Retroalimentación

El prototipo se ve muy bien y es suficientemente bueno.

Explicación de la Inclusión de Sugerencias en los Requisitos Funcionales

Las sugerencias y modificaciones a los prototipos fueron debidamente incorporadas dentro de la descripción de los requisitos funcionales. El uso de moldes básicos y estándar personalizables y recomendados según las medidas del cuerpo son parte esencial de los requisitos funcionales de personalización y previsualización de prendas, los cuales son fundamentales en el objetivo final del proyecto de proveer una muestra fidedigna de los prototipos para mejorar la eficiencia y claridad del proceso de diseño.

Además, el chatbot recomendado se ha incorporado como parte esencial del uso intuitivo y fluido del usuario. En particular es importante en el requisito no funcional de la usabilidad de la

plataforma, donde se espera que el usuario pueda completar sin complicaciones innecesarias tareas simples del proceso.

Historias Usuario

1. Diseñador de Moda Independiente

• Visualización de Diseños en 3D: Como diseñador independiente, quiero visualizar mis diseños en un modelo 3D realista, para reducir la cantidad de prototipos físicos y optimizar mi tiempo y materiales.

• Personalización de Prendas: Como diseñador independiente, quiero ajustar las medidas y estilos de mis diseños digitalmente, para ofrecer un servicio más eficiente y preciso a mis clientes.

• Optimización del Flujo de Trabajo: Como diseñador independiente, quiero una plataforma que almacene y organice mis diseños y medidas, para mejorar la gestión de mis proyectos.

• Colaboración con Clientes: Como diseñador independiente, quiero compartir avances de diseño de manera digital con mis clientes, para obtener su retroalimentación sin necesidad de reuniones presenciales.

2. Cliente Final con Interés en la Personalización

• Previsualización de Prenda en 3D: Como cliente, quiero ver una simulación digital de mi prenda en mi tipo de cuerpo, para asegurarme de que se ajuste correctamente antes de la confección.

• Personalización de Detalles: Como cliente, quiero poder elegir colores, telas y detalles de mi prenda a través de la plataforma, para reflejar mi estilo personal.

• Seguimiento del Pedido: Como cliente, quiero recibir actualizaciones en tiempo real sobre el estado de mi prenda, para sentirme seguro sobre los tiempos de entrega.

• Interacción con el Diseñador: Como cliente, quiero enviar comentarios sobre el diseño a través de la plataforma, para que mis sugerencias sean tomadas en cuenta antes de la producción final.

3. Diseñador con Poca Experiencia en Herramientas Digitales

• Interfaz Intuitiva: Como diseñador con poca experiencia digital, quiero una interfaz sencilla y amigable, para que pueda utilizar la herramienta sin necesidad de capacitación extensa.

• Tutoriales y Soporte: Como diseñador con poca experiencia digital, quiero acceso a tutoriales y guías paso a paso, para aprender a usar las herramientas sin complicaciones.

• Plantillas de Diseño: Como diseñador con poca experiencia digital, quiero contar con plantillas de diseño prediseñadas, para agilizar mi proceso de creación sin partir de cero.

• Simulación de Patronaje: Como diseñador con poca experiencia digital, quiero poder visualizar los patrones de mis diseños en un entorno digital, para reducir errores antes de la confección.

4. Administrador de la Plataforma

• Gestor de Usuarios: Como administrador, quiero gestionar los perfiles de diseñadores y clientes, para asegurar un uso adecuado de la plataforma.

• Control de Contenido: Como administrador, quiero moderar y verificar los diseños compartidos en la plataforma, para garantizar que cumplen con los estándares de calidad.

• Análisis de Uso: Como administrador, quiero acceder a estadísticas sobre el uso de la plataforma, para mejorar su rendimiento y funcionalidad.

Requisitos Funcionales

Registro e Inicio de Sesión para Usuarios

El sistema debe permitir a los usuarios, ya sean clientes finales, diseñadores de moda independientes o diseñadores con poca experiencia en herramientas digitales, crear cuentas personales y acceder a ellas para utilizar las funcionalidades del sistema. Esto incluye:

• Un formulario de registro para capturar información esencial como nombre, correo electrónico y contraseña.

• Verificación de correo electrónico para asegurar que la cuenta pertenece al usuario.

• Recuperación de contraseñas para ofrecer soporte en caso de olvido.

• Compatibilidad con opciones de inicio de sesión social (Google, Facebook).
• Inclusión de un *chatbot* asistente para facilitar el uso de la plataforma para usuarios nuevos que necesiten ayuda.

Historia de Usuario Enlazada

• Cliente Final: "Como cliente, quiero crear una cuenta donde pueda guardar mis preferencias de personalización y mi historial de pedidos para no repetir los mismos pasos en cada compra."

• Diseñador de Moda Independiente: "Como diseñador, necesito iniciar sesión en mi cuenta para ver y gestionar los diseños solicitados por los clientes, mantener un historial de ellos y seguir trabajando en proyectos previos."

• Diseñador con Poca Experiencia en Herramientas Digitales: "Como diseñador, prefiero que el registro sea simple y claro para poder comenzar a usar la plataforma rápidamente y sin complicaciones."

La autenticación asegura que los datos del usuario estén protegidos y que las funciones personalizadas (como guardar diseños o pedidos) estén disponibles solo para los usuarios registrados.

Personalización de Prendas

El sistema permitirá a los usuarios personalizar prendas según sus preferencias mediante una interfaz interactiva. Las opciones de personalización incluirán:

• Selección de tipos de prendas (camisas, vestidos, pantalones).

• Elección de telas, colores y patrones.

• Ajustes específicos de diseño, como longitudes, estilos de cuello, y tipos de botones.

• Capacidad para ingresar medidas personalizadas o seleccionar tamaños estándar.

Historia de Usuario Enlazada

• Cliente Final: "Como cliente, quiero personalizar las prendas seleccionando las telas y colores que reflejen mi estilo personal para asegurarme de que el producto final sea único."

• Diseñador de Moda Independiente: "Como diseñador, necesito ajustar digitalmente los patrones según las medidas que proporcionan los clientes para optimizar la precisión y reducir errores en la confección."

• Diseñador con Poca Experiencia en Herramientas Digitales: "Como diseñador, quiero opciones predefinidas de personalización para facilitar el proceso de ajuste y no cometer errores técnicos."

La personalización asegura que los usuarios puedan crear prendas que se adapten a sus gustos y necesidades específicas. Además, mejora la comunicación y el entendimiento entre diseñadores y clientes.

Previsualización de Prendas

El sistema debe ofrecer una herramienta para que los usuarios previsualicen sus prendas personalizadas en un modelo 3D o avatar. Esto incluye:

• Renderizado en 3D de la prenda con las opciones de personalización seleccionadas.
• Ajustes en tiempo real para reflejar cambios en las telas, colores o medidas.

• Capacidad de crear avatares personalizados basados en las medidas del usuario para ver cómo quedará la prenda en cuerpos reales.

Historia de Usuario Enlazada

• Cliente Final: "Como cliente, quiero una visualización 3D que me permita ver cómo se verá la prenda personalizada en un cuerpo similar al mío antes de realizar la compra."

• Diseñador de Moda Independiente: "Como diseñador, necesito una herramienta que me permita mostrar a mis clientes una representación precisa de cómo lucirá el diseño final, para reducir malentendidos o ajustes posteriores."

• Diseñador con Poca Experiencia en Herramientas Digitales: "Como diseñador, quiero previsualizar cómo quedará la prenda antes de iniciar la confección para asegurarme de que cumple con las expectativas del cliente."

La previsualización en 3D reduce el riesgo de errores y aumenta la confianza del cliente en el proceso, ofreciendo una experiencia inmersiva que mejora la calidad del servicio.

Seguimiento de Pedidos

El sistema debe permitir a los usuarios rastrear sus pedidos desde el momento de la personalización hasta la entrega. Las funciones incluirán:

• Actualizaciones automáticas del estado del pedido (e.g., "en producción", "en envío").
• Estimación del tiempo restante para la entrega.
• Notificaciones en caso de retrasos o problemas.
• Un historial de pedidos con detalles de cada uno.

Historia de Usuario Enlazada

• Cliente Final: "Como cliente, quiero poder rastrear el progreso de mi pedido para saber en qué etapa está y cuándo puedo esperarlo en mi domicilio."

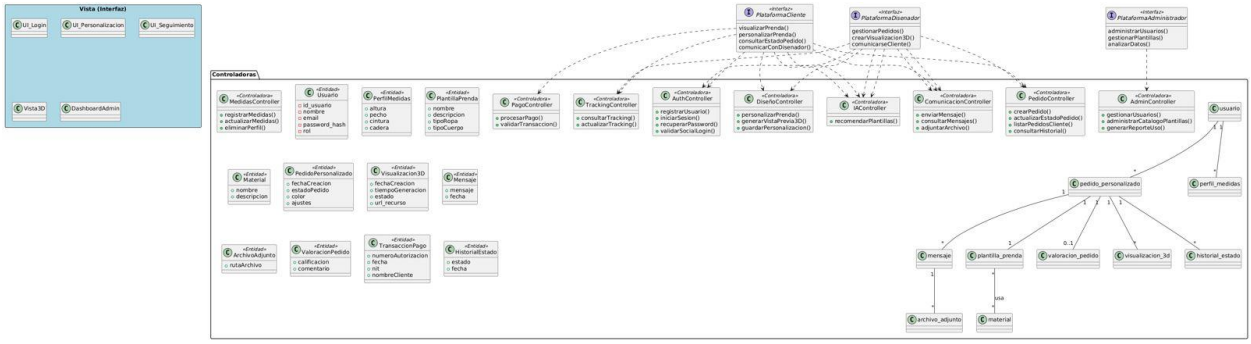
• Diseñador de Moda Independiente: "Como diseñador, necesito actualizar el estado de los pedidos en tiempo real para mantener informados a mis clientes y organizar mi flujo de trabajo."

• Diseñador con Poca Experiencia en Herramientas Digitales: "Como diseñador, quiero una función que me permita gestionar fácilmente los pedidos y asegurar que todo el proceso esté en orden."

El seguimiento en tiempo real aumenta la transparencia y confianza del cliente, mientras que ofrece a los diseñadores una mejor organización de sus procesos.

Clases Preliminares

Diagrama de Clases



Descripción de las Clases

Se realizó el diseño preliminar de las clases en base al modelo MVC que se estará implementando en el desarrollo de nuestro proyecto.

Interfaces (Vista)

- PlataformaCliente: Permite al cliente visualizar prendas en 3D, realizar personalizaciones de estilo y medidas, consultar el seguimiento del pedido y comunicarse directamente con el diseñador. Se conecta directamente con las controladoras para realizar estas funciones.
- PlataformaDisenador: Proporciona al diseñador herramientas para gestionar pedidos personalizados, visualizar y modificar diseños en 3D, comunicación con clientes, y controlar el tracking de pedidos, optimizando así la gestión de su trabajo.
- PlataformaAdministrador: Proporciona gestión de usuarios, contenidos, plantillas y estadísticas de uso. Conectada a controladoras administrativas específicas.

Clases Controladoras (Controladores)

- AuthController: Responsable de gestionar registros, autenticaciones, validaciones y sesiones (incluyendo login social). Interactúa directamente con usuarios.
- PedidoController: Gestiona el ciclo completo de pedidos personalizados desde su creación hasta cambios de estado y almacenamiento histórico.
- DiseñoController: Se enfoca en la generación de visualizaciones previas 3D, personalización de diseños (colores, materiales, ajustes) y almacenamiento seguro de estas personalizaciones.

- ComunicacionController: Controla el intercambio de mensajes y archivos entre diseñadores y clientes relacionados con los pedidos específicos.
- TrackingController: Maneja la consulta y actualización del estado de los pedidos, proporcionando información clara y transparente al cliente.
- IAController: Utiliza algoritmos inteligentes para recomendar plantillas según medidas corporales y preferencias de estilo del cliente, basándose en datos históricos.
- PagoController: Administra pagos, transacciones seguras, y asegura el cumplimiento legal (facturación SAT Guatemala).
- AdminController: Gestiona tareas administrativas: manejo de usuarios, pedidos, auditorías y estadísticas generales del sistema.
- MedidasController: Gestiona las medidas ingresadas por el usuario y se encarga de registrarlas y actualizarlas segun los datos que se indican.

Clases de Entidad (Modelo)

- Usuario: Representa a los usuarios del sistema (clientes, diseñadores o administradores). Contiene información para autenticación y perfil básico.
- PerfilMedidas: Almacena perfiles específicos de medidas corporales asociadas a clientes, facilitando personalizaciones exactas para cada usuario.
- PlantillaPrenda: Guarda modelos predefinidos de prendas, categorizadas por tipo de ropa y cuerpo, sirviendo como base para personalizaciones.
- Material: Almacena los diferentes materiales disponibles para ser aplicados a las prendas. Vinculado con plantillas para recomendar combinaciones.
- PedidoPersonalizado: Guarda información detallada de pedidos realizados por los clientes, incluyendo personalizaciones específicas, materiales, colores, medidas y el estado actual del pedido.
- Visualizacion3D: Registra visualizaciones previas generadas en 3D para cada pedido personalizado, incluyendo estados y tiempo de generación.
- Mensaje: Almacena mensajes intercambiados entre cliente y diseñador, facilitando la comunicación efectiva relacionada a cada pedido específico.
- ArchivoAdjunto: Guarda referencias a archivos adjuntos enviados dentro de los mensajes (imágenes, PDFs), facilitando la comunicación visual.
- HistorialEstado: Registra cambios en el estado del pedido personalizado, proporcionando trazabilidad clara del proceso desde creación hasta entrega.
- ValoracionPedido: Permite almacenar la valoración (calificación y comentario) de cada pedido finalizado, contribuyendo al feedback y mejora del servicio.
- TransaccionPago: Gestiona y almacena datos de transacciones de pago asociadas con cada pedido, cumpliendo con los requisitos fiscales y legales establecidos (SAT).

Relaciones Principales (Entidades)

- Usuario ↔ PerfilMedidas: Un usuario puede registrar múltiples perfiles de medidas personales o de terceros.

- Usuario ↔ PedidoPersonalizado: Un usuario puede realizar múltiples pedidos personalizados, cada uno asociado claramente a un solo usuario.
- PedidoPersonalizado ↔ PlantillaPrenda: Cada pedido personalizado se basa en exactamente una plantilla inicial seleccionada por el usuario.
- PlantillaPrenda ↔ Material (Muchos a Muchos): Cada plantilla puede tener múltiples materiales recomendados, y cada material puede estar recomendado en varias plantillas.
- PedidoPersonalizado ↔ Visualizacion3D: Cada pedido puede tener varias visualizaciones 3D asociadas, dependiendo de las iteraciones y ajustes realizados.
- PedidoPersonalizado ↔ HistorialEstado: Cada pedido personalizado tiene múltiples registros históricos de cambio de estado.
- Mensaje ↔ PedidoPersonalizado: Cada mensaje está vinculado a un pedido específico, facilitando la comunicación directa y contextualizada.
- ValoracionPedido ↔ PedidoPersonalizado: Cada pedido finalizado puede tener una valoración única.
- TransaccionPago ↔ PedidoPersonalizado: Cada pago se asocia específicamente a un pedido personalizado.

Casos de Uso y Clases Involucradas

Requisitos Funcionales

1. Registro e Inicio de Sesión para Usuarios

Casos de Uso

- Registrar Usuario
- Iniciar Sesión
- Recuperar Contraseña
- Autenticación Social

Clases Involucradas

- AuthController: Gestiona lógica de autenticación, validación y registro.
- Usuario: Almacena información y credenciales del usuario.
- PlataformaCliente, PlataformaDiseñador, PlataformaAdministrador: Interfaces para registrar y autenticar usuarios.

2. Personalización de Prendas

Casos de Uso

- Personalizar Prenda
- Seleccionar Material y Color
- Ingresar Medidas Personalizadas
- Guardar Personalización

Clases Involucradas

- DiseñoController: Controla personalizaciones de prendas, ajustando diseños y generando vistas previas.
- PlantillaPrenda: Base del diseño personalizado.
- Material: Materiales seleccionados para personalización.
- PerfilMedidas: Almacena medidas específicas usadas para ajustar patrones.
- MedidasController: gestionar el uso de las medidas del cuerpo para la personalización
- PedidoPersonalizado: Guarda personalización específica y detalles adicionales (color, ajustes, notas).
- PlataformaCliente, PlataformaDiseñador: Interfaces para realizar ajustes y personalizaciones.

3. Previsualización de Prendas en 3D

Casos de Uso

- Generar Visualización 3D
- Modificar Visualización en Tiempo Real
- Crear Avatar Personalizado

Clases Involucradas

- DiseñoController: Genera vistas previas en 3D basadas en ajustes.
- Visualizacion3D: Almacena resultados visuales generados, recursos digitales, estados y tiempos.
- PedidoPersonalizado: Proporciona datos específicos sobre la personalización a representar.
- PlataformaCliente, PlataformaDiseñador: Interfaces que muestran vistas previas 3D y permiten modificaciones.

4. Seguimiento de Pedidos

Casos de Uso

- Consultar Estado del Pedido
- Actualizar Estado del Pedido
- Notificar Cambios en Pedido
- Consultar Historial de Pedidos
- Comunicarse verbalmente acerca del Pedido

Clases Involucradas

- TrackingController: Gestiona actualizaciones en estados de pedidos, comunicación y notificaciones automáticas.
- PedidoController: Coordina el seguimiento, creación y gestión general de pedidos.
- HistorialEstado: Registra históricamente cambios de estado en pedidos.
- PedidoPersonalizado: Almacena el estado actual del pedido.
- ComunicaciónController: Gestiona la comunicación entre el cliente y el diseñador, permitiendo establecer cont acto directo y personal sobre el estado del pedido
- Mensaje: Representa los mensajes enviados en el flujo de la comunicación
- Archivo Adjunto: Respresenta algun archivo adjunto al mensaje enviado
- PlataformaCliente, PlataformaDiseñador: Interfaces para consultar estados actuales e históricos.

Requisitos No Funcionales

5. Confiabilidad (Disponibilidad del sistema 99%)

Casos de Uso

- Restaurar Sistema tras Caída
- Monitorear Estado del Sistema

Clases Involucradas

- AdminController: Gestiona monitoreo, recuperación y reinicio de servicios.
- PlataformaAdministrador: Interfaz para monitorear estados del sistema y realizar acciones de recuperación.

6. Usabilidad (Interfaz intuitiva)

Casos de Uso

- Navegación Simplificada en Plataformas
- Uso de Ayudas y Tutoriales

Clases Involucradas

- PlataformaCliente, PlataformaDiseñador, PlataformaAdministrador: Interfaces diseñadas con alta usabilidad.
- IAController: Asistente inteligente que recomienda opciones basadas en interacciones previas del usuario.

7. Portabilidad

Casos de Uso

- Despliegue del Sistema en Distintas Plataformas (Linux, Windows, macOS)

Clases Involucradas

- AdminController: Configura y asegura que la aplicación pueda correr en distintos entornos.
- Indirectamente todas las clases, asegurando que su código fuente es compatible en distintos sistemas operativos.

8. Cumplimiento Legal (Facturación SAT Guatemala)

Casos de Uso

- Generar Factura Electrónica
- Validar Cumplimiento Fiscal

Clases Involucradas

- PagoController: Gestiona creación y validación de transacciones y facturas electrónicas.
- TransaccionPago: Información estructurada y almacenada para cumplimiento fiscal.

9. Escalabilidad

Casos de Uso

- Escalar Sistema Según Demanda de Usuarios
- Monitorear Rendimiento bajo Alta Demanda

Clases Involucradas

- AdminController: Monitorea rendimiento del sistema y gestiona ajustes de escalabilidad técnica.
- Todas las entidades persistentes involucradas en accesos concurrentes (Usuario, PedidoPersonalizado, etc).

10. Ética (No almacenamiento datos sensibles)

Casos de Uso

- Validar Transacción sin Almacenamiento de Información Sensible

Clases Involucradas

- PagoController: Responsable de garantizar que información sensible (tarjetas de crédito) no se almacene en la base de datos.

11. Rendimiento (Vista previa generada en ≤ 4 segundos)

Casos de Uso

- Generar Visualización 3D Eficientemente

Clases Involucradas

- DiseñoController: Garantiza generación rápida de vistas previas 3D.
- Visualizacion3D: Almacena tiempos de generación para monitoreo.

12. Mantenibilidad

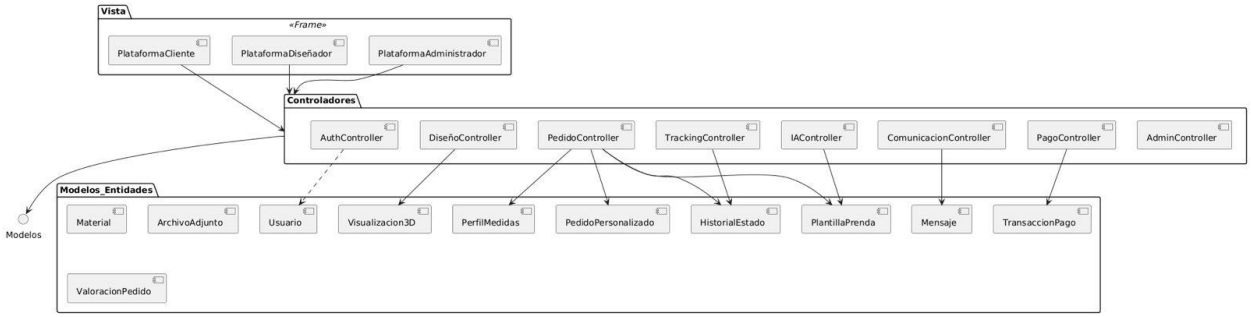
Casos de Uso

- Actualizar Plantillas y Diseños Rápidamente

Clases Involucradas

- AdminController: Gestión eficiente de actualización en plantillas.
- PlantillaPrenda y Material: Permiten modificaciones rápidas, estructura simple y clara.

Diagrama de Paquetes



Descripción de los Paquetes y sus Componentes

Vista (Interfaz)

Este paquete agrupa todas las interfaces gráficas y componentes que permiten la interacción directa con los usuarios del sistema (clientes, diseñadores y administradores). Aquí se encuentran:

- PlataformaCliente: Interfaz utilizada por clientes finales para visualizar prendas en 3D, realizar personalizaciones, consultar estados de pedidos, pagos y comunicarse con diseñadores.
- PlataformaDiseñador: Herramienta interactiva utilizada por los diseñadores para gestionar pedidos, visualizar personalizaciones, comunicarse con clientes, y administrar el proceso de creación y producción.
- PlataformaAdministrador: Herramienta utilizada por los administradores para gestionar usuarios, contenidos, revisar estadísticas de uso y mantener la plataforma segura y funcional.

Controladores

Paquete que agrupa las clases responsables de la lógica de negocio y la coordinación entre las interfaces (vistas) y los datos (entidades). Controlan operaciones como autenticación, gestión de pedidos, seguimiento, comunicación y recomendaciones con inteligencia artificial. Incluye:

- AuthController: Gestiona autenticación, registros, recuperación de contraseñas, validación de accesos mediante redes sociales.
- PedidoController: Responsable del ciclo de vida del pedido personalizado (creación, actualización de estados y gestión general).
- DiseñoController: Encargado de la personalización detallada de prendas, generación de vistas previas en 3D, y almacenamiento de ajustes específicos.
- ComunicacionController: Administra mensajes y archivos adjuntos entre usuarios para mejorar la comunicación y evitar malentendidos.
- TrackingController: Gestiona actualizaciones sobre el estado del pedido, seguimiento en tiempo real y proporciona claridad en tiempos estimados.
- IAController: Utiliza algoritmos inteligentes para recomendar diseños de prendas personalizadas basadas en perfiles corporales específicos.

- PedidoController: Centraliza la creación, modificación y administración general de los pedidos personalizados realizados por clientes.
- PagoController: Gestiona transacciones financieras y asegura cumplimiento legal con normativas de facturación (SAT Guatemala).

Entidades (Modelo)

Este paquete agrupa las clases que representan los datos principales del sistema, incluyendo la información de usuarios, prendas, pedidos, comunicación y pagos. Estas clases interactúan con la base de datos directamente, correspondiendo a las tablas relacionales definidas previamente. Contiene:

- Usuario: Almacena información básica y detalles de roles (clientes, diseñadores, administradores).
- PerfilMedidas: Representa diferentes perfiles con medidas específicas para personalizar prendas de clientes individuales.
- PlantillaPrenda: Modelos base de prendas que pueden ser personalizados por los usuarios.
- Material: Información sobre los materiales y telas utilizados en las prendas.
- PedidoPersonalizado: Información completa de pedidos personalizados realizados por clientes, incluyendo ajustes, materiales, colores y estado.
- Visualizacion3D: Registros sobre representaciones 3D generadas por el sistema para previsualización de prendas.
- HistorialEstado: Histórico detallado de cambios de estado en los pedidos para seguimiento y transparencia.
- Mensaje: Mensajes intercambiados entre clientes y diseñadores para facilitar la comunicación sobre los pedidos.
- ArchivoAdjunto: Archivos adicionales que acompañan a mensajes entre usuarios.
- ValoracionPedido: Comentarios y calificaciones finales de los clientes sobre los pedidos finalizados.
- TransaccionPago: Registro detallado de pagos realizados, asegurando cumplimiento fiscal y transparencia contable.

Relaciones entre Paquetes

- Interfaces (Vista) dependen directamente de las Controladoras, ya que invocan sus métodos para responder a las acciones de usuario.
- Controladoras interactúan directamente con Entidades (datos del Modelo) para acceder, modificar y persistir información en la base de datos.

Persistencia de Datos

Diagrama de Clases Persistentes

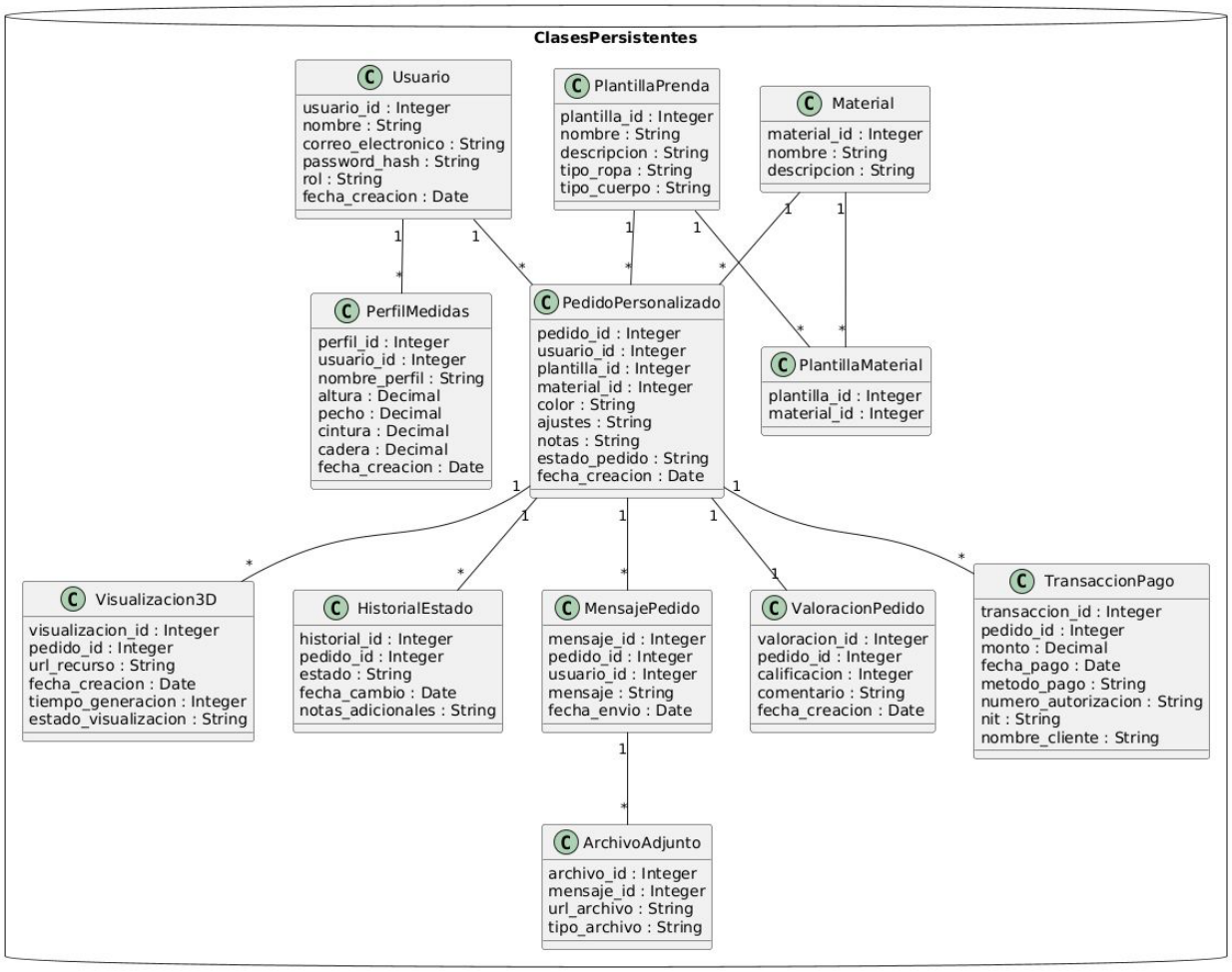
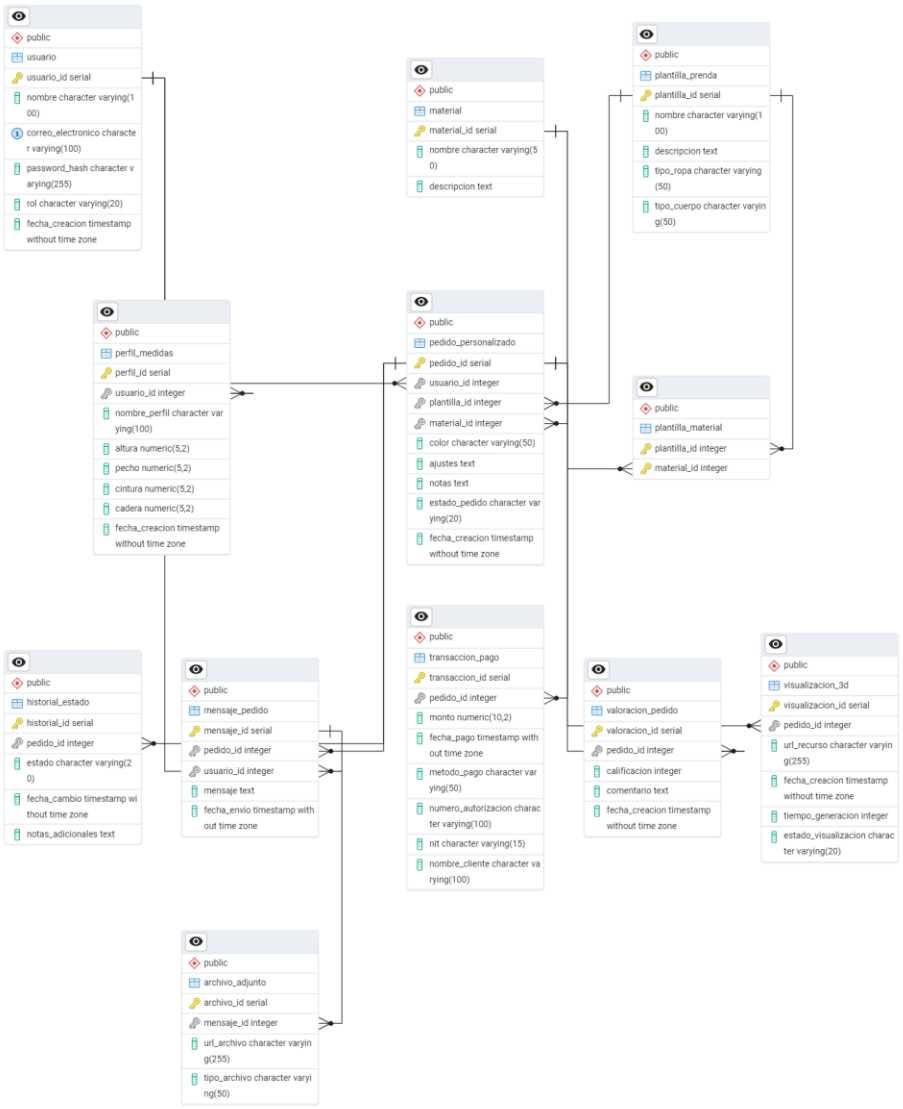


Diagrama Entidad Relación



Estimaciones Requisitos No Funcionales

Confiabilidad

1. El sistema debe garantizar una disponibilidad mínima del 99% en horario laboral para diseñadores y clientes. Se monitoreará el tiempo de inactividad semanal y se calculará el porcentaje de disponibilidad.

2. En caso de falla del servidor o pérdida de conectividad, el sistema debe reestablecer sus servicios críticos (carga de proyectos, modelado 3D y acceso a catálogos) en un máximo de 5 minutos. Se realizarán simulaciones de caída de servidor y se medirá el tiempo hasta restablecer la funcionalidad esencial.

Dimensiones:

- Infraestructura: Requiere servidores de alta disponibilidad, balanceadores de carga, backups automáticos, y monitoreo constante.
- Herramientas necesarias: Servicios en la nube como AWS EC2, Azure App Services, o Google Cloud. Sistemas de monitoreo como Datadog o New Relic.
- Escala en infraestructura: Dos servidores activos para redundancia + al menos un sistema de recuperación de emergencia (backup en caliente).
- Tiempo estimado de implementación: 2-3 meses para configurar, probar y validar la disponibilidad y tiempos de restauración en simulaciones.

Usabilidad

3. La plataforma debe contar con una interfaz intuitiva. Al realizar pruebas de usabilidad con al menos cinco usuarios representativos, la tasa de errores que impidan completar una tarea durante la navegación inicial no debe superar el 10%. Se medirá mediante heurísticas de usabilidad y conteo de errores críticos.

Dimensiones:

- Pruebas: Realizar al menos 2 ciclos de pruebas con grupos de 5-10 usuarios representativos.
- Tiempo dedicado: 2-3 semanas por ciclo de prueba incluyendo ajustes con base en el feedback.
- Herramientas necesarias: Prototipos interactivos (usando Figma o Adobe XD) y herramientas de pruebas de usabilidad como Hotjar o UserTesting.
- Recursos humanos: Diseñador UI/UX + moderadores para supervisar las pruebas.
- Escala del proceso: Al menos 10 sesiones de prueba, ajustes y validaciones finales.

Portabilidad

4. El sistema debe poder trasladarse y ejecutarse en diferentes entornos, incluyendo Windows, macOS y Linux. Se realizarán pruebas en distintos entornos de ejecución, asegurando que al menos el 90% de los entornos probados permitan la instalación y ejecución sin modificaciones significativas.

Dimensiones:

- Entornos de prueba: Versiones recientes de Windows 10/11, macOS Ventura/Monterey, y Ubuntu 20.04 o Fedora.
- Herramientas: Implementación multiplataforma con Docker o entornos virtualizados para pruebas.
- Tiempo dedicado: 3 semanas para pruebas en los tres sistemas operativos, ajustando dependencias según el entorno.
- Escala: Soporte probado en 90% de los entornos clave con configuraciones estándar.

Cumplimiento Legal y Normativo

5. La generación de facturas electrónicas debe cumplir con las normativas SAT de Guatemala. Se validará automáticamente con el sistema SAT, realizando pruebas de certificación, auditorías trimestrales sin incumplimientos y asegurando el almacenamiento de facturas por 5 años.

6. Toda imagen o plantilla utilizada en la plataforma debe contar con permisos y licencias adecuadas (libre uso, creative commons o contrato comercial). El 100% de los recursos deberán tener registros de fuente y tipo de licencia en el repositorio documental. Se auditarán los catálogos trimestralmente.

Dimensiones:

- Integración SAT: 2-3 semanas para desarrollo e integración de sistemas de facturación compatibles con la API del SAT.
- Auditorías de imágenes: Gestión de repositorios documentados con herramientas como Google Drive o sistemas de archivo dedicados (e.g., DocuWare).
- Tiempo dedicado: 1 semana por auditoría trimestral; hasta 2 meses iniciales para establecer estándares y procesos de licencia.
- Escala del proceso: Certificaciones iniciales y revisiones periódicas para mantener la conformidad.

Escalabilidad

7. El sistema debe manejar incrementos en la cantidad de usuarios sin degradar el rendimiento. Se realizarán pruebas incrementando progresivamente la carga de usuarios y monitoreando el rendimiento, asegurando que el sistema funcione correctamente con hasta 10 veces la carga esperada.

Dimensiones:

- Entornos de prueba: Versiones recientes de Windows 10/11, macOS Ventura/Monterey, y Ubuntu 20.04 o Fedora.
- Herramientas: Implementación multiplataforma con Docker o entornos virtualizados para pruebas.
- Tiempo dedicado: 3 semanas para pruebas en los tres sistemas operativos, ajustando dependencias según el entorno.
- Escala: Soporte probado en 90% de los entornos clave con configuraciones estándar.

Ética

8. El sistema de pago con tarjeta de débito/crédito no debe almacenar datos ingresados por el usuario. Se analizarán logs y bases de datos para detectar almacenamiento no autorizado, se realizarán pruebas de penetración para identificar vulnerabilidades y se certificarán los sistemas sin persistencia de información sensible post transacción.

Dimensiones:

- Integración con pasarelas de pago: Uso de Stripe, PayPal o una alternativa que maneje la seguridad del lado del proveedor.
- Pruebas: Pruebas de penetración trimestrales con herramientas como OWASP ZAP o Burp Suite para garantizar que no se almacenan datos sensibles.
- Tiempo dedicado: 2 semanas para configurar integraciones y pruebas iniciales.
- Escala del proceso: Auditorías de seguridad continuas cada trimestre.

Rendimiento

9. La plataforma debe generar la vista previa 3D de un diseño (cuerpo + prenda) en un máximo de 4 segundos para plantillas de complejidad media (moldes con hasta 3 piezas principales). Se medirán tiempos promedio de renderizado en pruebas de carga con 10 usuarios concurrentes.

Dimensiones:

- Motor de renderizado: Herramientas como Three.js o Babylon.js optimizadas para modelos ligeros.
- Hardware del servidor: GPU o servicios de renderizado en la nube como AWS EC2 con GPU integrada.
- Pruebas: Pruebas de tiempo con cargas de 10, 50 y 100 usuarios concurrentes.
- Tiempo dedicado: 4-8 semanas para implementación y optimización del renderizado 3D.
- Escala del proceso: Capacidad para manejar plantillas de 3 piezas principales en cargas medias.

Mantenibilidad

10. El sistema debe permitir agregar o actualizar una plantilla o molde nuevo en un tiempo máximo de 3 horas, empleando procedimientos documentados. Se medirá el tiempo requerido para que un administrador capacitado complete el proceso y valide su correcto funcionamiento.

Dimensiones:

- Procedimientos: Documentación clara y entrenamiento para administradores.
- Herramientas: Interfaces intuitivas de administración como Django Admin o paneles personalizados.
- Tiempo dedicado: 1 mes inicial para diseñar el flujo de gestión y documentarlo.
- Escala del proceso: Proceso simplificado con pasos claros para cambios regulares.

Selección de Tecnología Frontend

Tecnologías Consideradas

Para la implementación del frontend, se han evaluado las siguientes opciones:

- Angular: Es un framework desarrollado por Google que utiliza TypeScript y sigue una arquitectura basada en componentes (Google Developers, 2023).
  Ventajas:
  - Ofrece una solución completa con herramientas integradas para manejo de estado, enrutamiento, validación de formularios y más.
  - Posee una estructura bien definida que facilita el desarrollo de aplicaciones grandes y complejas.

  Desventajas:
  - Tiene una curva de aprendizaje alta, lo que puede ralentizar el desarrollo inicial.
  - El binding bidireccional, si bien es potente, puede afectar el rendimiento en aplicaciones muy interactivas o de alta carga.
- Vue.js:
  Es un framework progresivo y ligero que se destaca por su sistema reactivo y su facilidad de integración en proyectos existentes (You, 2023).
  Ventajas:
  - Su sintaxis es sencilla y fácil de aprender, lo que permite un desarrollo rápido y eficiente.
  - Su ecosistema flexible facilita la integración con otras bibliotecas o herramientas.

  Desventajas:
  - La comunidad, aunque en crecimiento, es menor en comparación con Angular o React, lo que puede limitar el soporte y la disponibilidad de recursos en proyectos de gran envergadura.
  - En aplicaciones muy grandes, sin una arquitectura bien definida, pueden surgir problemas de escalabilidad y mantenimiento.
- React.js:
  Es una biblioteca desarrollada por Meta (Facebook) para la creación de interfaces de usuario, basada en componentes y en el uso del Virtual DOM (Meta Open Source, 2023).
  Ventajas:
  - Altamente flexible y eficiente en el renderizado, permitiendo una excelente experiencia de usuario.
  - Cuenta con un extenso ecosistema de herramientas (por ejemplo, Redux para gestión del estado) y una gran comunidad, lo que facilita encontrar soluciones a problemas comunes.

  Desventajas:
  - Al ser solo la capa de vista, requiere la integración con otras librerías para lograr funcionalidades completas como el enrutamiento o el manejo de datos en el lado del servidor, lo que puede incrementar la complejidad de la configuración inicial.

- Next.js:
  Es un framework construido sobre React que incorpora características avanzadas como renderizado del lado del servidor (SSR), generación de sitios estáticos (SSG) y soporte para API routes (Vercel, 2023).

  Ventajas:
  - Optimiza el rendimiento y la carga inicial gracias a sus técnicas de pre-renderizado, lo cual es crucial para cumplir con requisitos de rapidez en la visualización (por ejemplo, vistas 3D en menos de 4 segundos).
  - Mejora la escalabilidad y disponibilidad, facilitando la replicación y distribución de la información en la nube.
  - Aumenta la visibilidad en motores de búsqueda (SEO) gracias a su capacidad de generar contenido pre-renderizado.

  Desventajas:

  - Puede introducir una complejidad adicional en la configuración inicial, especialmente para desarrolladores menos familiarizados con conceptos como SSR y SSG.
  - Requiere una buena planificación en el manejo de rutas y datos, aunque esta complejidad se compensa con las ventajas en rendimiento y escalabilidad.

Evaluación Basada en Características del Framework y Estimaciones de los Requisitos
El proyecto debe cumplir con los siguientes criterios clave:

2.1 Escalabilidad

El sistema debe manejar un incremento en la cantidad de usuarios sin degradar el rendimiento. Next.js ofrece SSR y SSG, lo que optimiza la carga de datos y mejora la escalabilidad sin recargar el servidor.

2.2 Rendimiento

La plataforma debe generar vistas previas en 3D en un tiempo máximo de 4 segundos. Next.js mejora el rendimiento mediante su capacidad de pre-renderizado, reduciendo el tiempo de carga en comparación con soluciones SPA como Vue.js o React sin SSR.

2.3 Confiabilidad y Disponibilidad

El sistema debe garantizar una disponibilidad del 99% en horario laboral. Next.js facilita la replicación de información y recuperación rápida con su arquitectura optimizada para distribución en la nube.

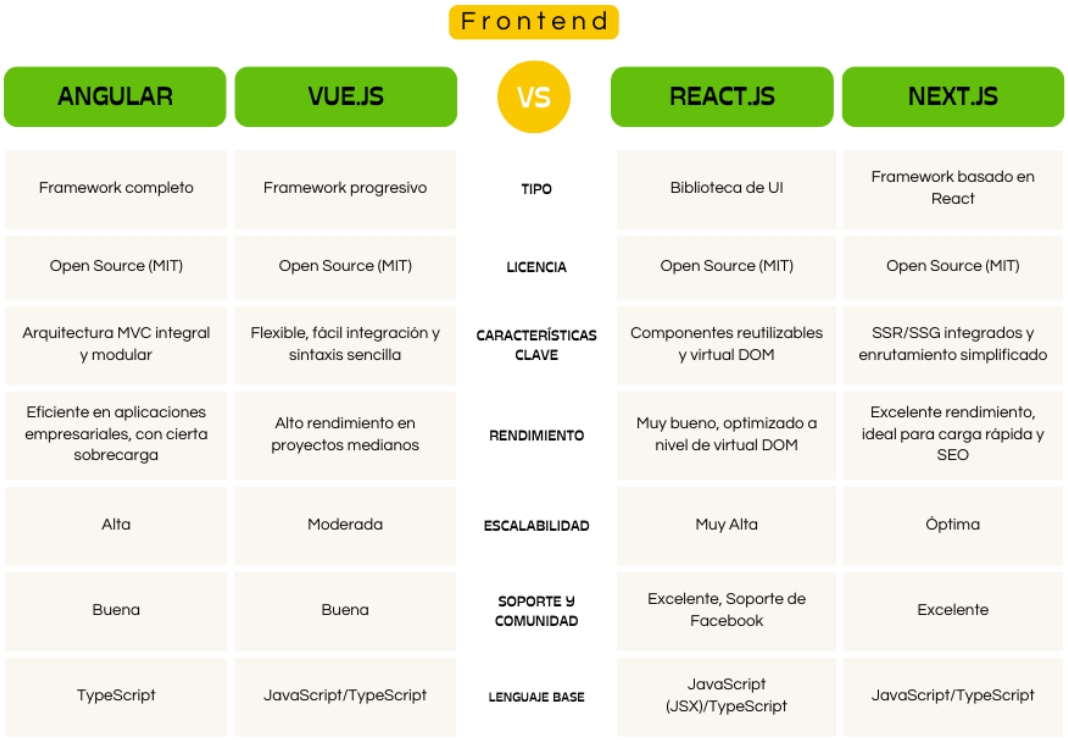
2.4 Mantenibilidad

El sistema debe permitir agregar o actualizar plantillas en un tiempo máximo de 3 horas. La modularidad de React y Next.js permite una estructura escalable y de fácil mantenimiento.

2.5 Usabilidad y Portabilidad

Se requiere una interfaz intuitiva accesible desde diferentes entornos. React y Next.js permiten crear interfaces responsivas optimizadas para web y móvil.

Comparación de Tecnologías



Justificación de la Elección

La combinación de React.js y Next.js se destaca frente a las demás tecnologías evaluadas por las siguientes razones:

1. Ventajas en Rendimiento y Escalabilidad:
    a. Next.js aprovecha el renderizado del lado del servidor y la generación estática, lo que permite tiempos de carga significativamente menores. Esto es crucial para cumplir con los exigentes tiempos de respuesta (menos de 4 segundos para vistas 3D) y para gestionar incrementos en la carga de usuarios sin afectar la experiencia del usuario.

   b. Frente a Angular y Vue.js, que dependen principalmente del procesamiento en el cliente, la arquitectura de Next.js ofrece una mejor gestión de recursos en aplicaciones de gran escala.
2. Flexibilidad y Ecosistema:
   a. React.js brinda una gran flexibilidad para construir interfaces de usuario a través de un sistema basado en componentes, lo que facilita el mantenimiento y la ampliación del sistema.
   b. La extensa comunidad y el ecosistema robusto de React permiten la integración de librerías y herramientas complementarias (por ejemplo, para gestión del estado o enrutamiento), superando a las soluciones que pueden resultar menos modulares o más rígidas en su estructura.
3. Optimización SEO y Experiencia de Usuario:
   a. La capacidad de Next.js para generar contenido pre-renderizado mejora la indexación en motores de búsqueda, lo cual es un valor agregado para cualquier plataforma digital.
   b. La experiencia de usuario se beneficia de interfaces interactivas y rápidas, esenciales para la personalización en tiempo real, algo en lo que React brilla gracias a su Virtual DOM.
4. Mantenibilidad y Desarrollo Ágil:
   a. La estructura modular de React junto con las características avanzadas de Next.js permite que nuevas funcionalidades y actualizaciones (como la incorporación de nuevas plantillas de diseño) se realicen de forma ágil y con menos riesgo de introducir errores, a diferencia de Angular, cuya complejidad puede ralentizar iteraciones de desarrollo.
   b. Aunque Vue.js presenta una curva de aprendizaje baja y buena flexibilidad, en comparación, la combinación de React y Next.js ofrece una solución más robusta y escalable para proyectos de alto rendimiento y disponibilidad.

Selección de Tecnología Backend

Tecnologías Consideradas

Para la implementación del backend se han evaluado las siguientes opciones, considerando aspectos como lenguaje, arquitectura, rendimiento, escalabilidad, facilidad de uso, seguridad y casos de uso:

A) Django:

Es un framework de alto nivel escrito en Python, diseñado para el desarrollo rápido y seguro de aplicaciones web (Holovaty & Kaplan-Moss, 2023). Ventajas:

- Desarrollo rápido y sencillo gracias a la filosofía “baterías incluidas”.
- Código limpio y mantenible, favorecido por la claridad y legibilidad de Python.
- Medidas de seguridad robustas integradas por defecto (protección contra CSRF, XSS, inyección SQL, etc.).

- Amplio ecosistema y documentación, lo que facilita la solución de problemas y la incorporación de nuevas funcionalidades.
- Escalabilidad adecuada para aplicaciones de mediana a gran escala, con buenas opciones de integración para caché y despliegue en la nube.

Desventajas:

- Puede requerir optimizaciones adicionales en escenarios de alta concurrencia.
- En proyectos extremadamente complejos, la estructura "baterías incluidas" puede resultar menos flexible en comparación con soluciones más modulares.

B) Ruby on Rails:
Es un framework basado en Ruby que enfatiza la convención sobre la configuración y la rapidez en el desarrollo.
Ventajas:

- Filosofía "convención sobre configuración" que acelera el desarrollo inicial.
- Código conciso y fácil de entender, lo que puede mejorar la productividad.
- Buenas medidas de seguridad integradas y una arquitectura que favorece la organización del código.

Desventajas:

- Desafíos en escalabilidad para aplicaciones de gran envergadura, requiriendo ajustes y estrategias adicionales.
- Menor flexibilidad en personalizaciones complejas debido a la fuerte dependencia en convenciones predefinidas.

C) Laravel:
Es un framework de PHP que se destaca por su elegante sintaxis y su enfoque en la simplicidad y la productividad (Otwell, 2023).
Ventajas:

- Sintaxis elegante y moderna que permite un desarrollo ágil.
- Gran comunidad y abundante documentación que facilitan el aprendizaje y la resolución de incidencias.
- Integración de múltiples herramientas que ayudan en tareas comunes como autenticación, enrutamiento y manejo de bases de datos.

Desventajas:

- Puede presentar limitaciones de rendimiento en escenarios de alta concurrencia si no se implementan correctamente estrategias de optimización y caché.
- La flexibilidad y potencia de PHP pueden depender en gran medida de la experiencia del desarrollador, lo que puede afectar la robustez del sistema.

D) Spring Boot (o Spring Root):
Es un framework basado en Java que simplifica la creación de aplicaciones independientes y productivas (Walls, 2022).
Ventajas:

- Altísimo rendimiento y solidez, especialmente en entornos empresariales de alta carga.

- Excelente escalabilidad, ideal para arquitecturas basadas en microservicios y aplicaciones de misión crítica.
- Amplias características de seguridad integradas, probadas en numerosos entornos críticos.

Desventajas:

- Curva de aprendizaje más pronunciada, debido a la complejidad inherente del ecosistema Java.
- Requiere una mayor inversión inicial en configuración y despliegue, lo que puede ralentizar el desarrollo en fases tempranas.

Evaluación Basada en Características del Framework y Estimaciones de los Requisitos

Para tomar una decisión informada, se han considerado los siguientes aspectos:

- Lenguaje: Python (Django) se destaca por su simplicidad y legibilidad, lo que favorece el mantenimiento y la colaboración.
- Arquitectura: Django utiliza el patrón MVT, ofreciendo una estructura clara y modular, mientras que las otras opciones utilizan MVC o arquitecturas basadas en microservicios.
- Rendimiento: Aunque Spring Boot ofrece un rendimiento excelente en entornos empresariales, Django se adapta de forma equilibrada a proyectos de mediana y gran escala sin complejidades excesivas.
- Escalabilidad: Django, con sus capacidades de caché y soporte para despliegue en la nube, permite escalar la aplicación de forma eficiente.
- Facilidad de Uso: La curva de aprendizaje de Django es menor en comparación con Spring Boot y ofrece una sintaxis limpia y una gran cantidad de recursos y documentación.
- Seguridad: Django incorpora de forma nativa medidas de seguridad robustas, lo que es esencial para proteger la información sensible de la plataforma.
- Casos de Uso: Para una plataforma que requiere rapidez en el desarrollo, mantenimiento ágil y alta seguridad, Django se alinea perfectamente con los requisitos del proyecto.

Comparación de Tecnologías

CRITERIO	DJANGO	RUBY ON RAILS	LARAVEL	SPRING BOOT
Lenguaje	Python	Ruby	PHP	Java
Arquitectura	Model-Template-View (MTV)	Model-View-Controller (MVC)	Model-View-Controller (MVC)	Model-View-Controller (MVC)
Rendimiento	Adecuado para aplicaciones con lógica compleja, pero requiere optimización para alto tráfico	Rendimiento decente, pero Ruby puede ser más lento en comparación con otros frameworks debido a su naturaleza interpretada	Rendimiento moderado, adecuado para aplicaciones pequeñas y medianas	Alto rendimiento, especialmente para aplicaciones empresariales complejas
Escalabilidad	Esscalable, pero puede requerir optimización para manejar grandes cargas	Escalable para proyectos medianos, pero puede ser un desafío en aplicaciones muy grandes o de tráfico intenso	Escalable, pero no tan eficiente como Django o Spring Boot para grandes cargas	Muy escalable, diseñado para apliaciones empresariales de gran escala
Facilidad de uso	Fácil de aprender y usar, especialmente para principiantes en desarrollo web	Intuitivo para desarrolladores familiarizado con Ruby, excelente para construir prototipos rápidamente	Fácil de usar para desarrolladores familiarizados con PHP	Más complejo, requiere experiencia en Java y su ecosistema
Seguridad	Alto nivel de seguridad con características integradas como protección contra CSFR, XSS y SQL Injection	Ofrece medidas de seguridad integradas, pero necesita configuraciones adicionales para ciertos ataques	Proporciona características de seguridad integradas, pero requiere configuración adicional	Muy seguro, con soporte para estándares empresariales de seguridad
Casos de Uso	Ideal para aplicaciones con lógica compleja, paneles de administración y APIs REST	Ideal para startups y proyectos donde la velocidad de desarrollo es clave	Adecuado para aplicaciones pequeñas y medianas, como blogs y sitios web	Perfecto para aplicaciones empresariales y sistemas distribuidos

Justificación de la Elección

Tras evaluar las ventajas y desventajas de cada tecnología en función de los criterios establecidos, se opta por Django por las siguientes razones:

- Lenguaje y Facilidad de Uso: Python es ampliamente reconocido por su simplicidad y legibilidad, lo que facilita el mantenimiento del código y permite que equipos de desarrollo colaboren de forma eficiente. Django, al estar basado en Python, aprovecha estas ventajas, reduciendo la curva de aprendizaje y acelerando el proceso de desarrollo.
- Arquitectura y Mantenibilidad: El patrón Model-View-Template (MVT) que utiliza Django ofrece una clara separación de responsabilidades, lo que facilita la organización y el mantenimiento del proyecto a largo plazo. Esta modularidad permite implementar y actualizar funcionalidades (como la integración de nuevas herramientas o módulos de seguridad) de manera ágil.
- Rendimiento y Escalabilidad: Django ofrece un rendimiento sólido adecuado para aplicaciones de mediana a gran escala, y su capacidad para integrarse con sistemas de caché y despliegue en la nube favorece una escalabilidad eficaz.

  Aunque frameworks como Spring Boot pueden ofrecer un rendimiento superior en entornos empresariales, la complejidad adicional y la mayor curva de aprendizaje no se justifican en este contexto, donde la agilidad y facilidad de desarrollo son prioritarias.

- Seguridad: La seguridad es un aspecto crítico en esta plataforma, y Django incorpora de forma nativa medidas contra vulnerabilidades comunes (como CSRF, XSS e inyección SQL), lo que garantiza una protección robusta sin necesidad de configuraciones excesivas.
- Casos de Uso y Flexibilidad: Django se adapta muy bien a los requerimientos de una plataforma que necesita desarrollo rápido, alta fiabilidad y facilidad para implementar cambios en función del crecimiento del sistema y las necesidades del negocio.
  Su amplio ecosistema y comunidad aseguran que existen soluciones y recursos para prácticamente cualquier necesidad que surja durante el ciclo de vida del proyecto.

Selección de Tecnología Persistencia de Datos

Tecnologías Consideradas

Para la persistencia de datos se han evaluado las siguientes opciones, analizando aspectos clave como integridad de datos, rendimiento, escalabilidad, facilidad de uso, seguridad y casos de uso:

A) PostgreSQL:

Es un sistema de gestión de bases de datos relacional (RDBMS) open-source que destaca por su robustez y capacidades avanzadas.

Ventajas:

◊ Alta integridad de datos gracias a las transacciones ACID.
◊ Excelente soporte para tipos de datos complejos (JSON, arreglos, tipos geométricos).
◊ Alto rendimiento mediante el uso de índices avanzados y optimización para consultas complejas.
◊ Gran capacidad de escalabilidad vertical y moderada escalabilidad horizontal.
◊ Amplia comunidad y abundante documentación disponible.
◊ Herramientas avanzadas, como triggers, funciones y procedimientos almacenados.

Desventajas:

◊ Requiere una configuración y mantenimiento adecuados para obtener el mejor rendimiento.
◊ Puede resultar inicialmente más complejo en comparación con soluciones más sencillas como SQLite.

B) MySQL:

Es uno de los sistemas de gestión de bases de datos más populares, ampliamente utilizado en proyectos web.

Ventajas:

◊ Excelente rendimiento en transacciones básicas y rápidas.
◊ Gran facilidad de configuración y administración.
◊ Herramientas robustas disponibles (MySQL Workbench, phpMyAdmin).
◊ Buena documentación y soporte de la comunidad.
◊ Uso extendido en múltiples proyectos web.

Desventajas:

◊ Menor soporte para tipos de datos avanzados y estructuras complejas.
◊ Escalabilidad limitada en consultas muy complejas.
◊ Menor flexibilidad comparado con PostgreSQL en consultas avanzadas.

- ◊ Algunas funcionalidades avanzadas pueden requerir versiones Enterprise o plugins comerciales.

C) SQLite:

Es un motor de bases de datos ligero que se integra directamente en la aplicación.

Ventajas:

- ◊ Muy ligera y sencilla de implementar.
- ◊ Ideal para prototipos rápidos o aplicaciones móviles/desktop de pequeño tamaño.
- ◊ No requiere infraestructura compleja.

Desventajas:

- ◊ Limitaciones importantes en rendimiento con grandes volúmenes de datos o usuarios concurrentes.
- ◊ Poco recomendable para aplicaciones con múltiples usuarios simultáneos.
- ◊ Baja escalabilidad y menor integridad en comparación con sistemas más robustos.

D) SQL Server (Relational):

Es una solución de bases de datos desarrollada por Microsoft, reconocida por su robustez en entornos empresariales.

Ventajas:

- ◊ Rendimiento excepcional en transacciones complejas y manejo de alto volumen de datos.
- ◊ Excelente integridad de datos mediante soporte completo de transacciones (ACID).
- ◊ Soporte robusto para tipos de datos complejos (JSON, XML).
- ◊ Herramientas gráficas completas (SQL Server Management Studio).
- ◊ Alta integración con tecnologías del ecosistema Microsoft (Azure, Power BI, .NET).
- ◊ Muy buena seguridad con auditorías avanzadas y controles estrictos de acceso.

Desventajas:

- ◊ Altos costos asociados a licencias comerciales, especialmente en funcionalidades avanzadas.
- ◊ Limitada escalabilidad horizontal sin configuraciones especializadas.
- ◊ Complejidad inicial y de mantenimiento relativamente alta.
- ◊ Dependencia del entorno Microsoft, lo que puede limitar la integración con soluciones open-source.

Evaluación Basada en Características del Framework y Estimaciones de los Requisitos

Para tomar una decisión informada, se han analizado los siguientes aspectos:

- Integridad de Datos:
  PostgreSQL y SQL Server sobresalen gracias a su robusto cumplimiento de transacciones ACID, mientras que MySQL y SQLite pueden presentar limitaciones en escenarios de datos complejos.
- Rendimiento:
  PostgreSQL ofrece un rendimiento alto en consultas complejas mediante el uso de índices avanzados, en tanto que MySQL brinda un excelente desempeño en transacciones básicas. SQLite es ideal para aplicaciones ligeras, y SQL Server se orienta a entornos de alta demanda, aunque con mayor costo y complejidad.

- Escalabilidad:
  PostgreSQL permite escalabilidad vertical y moderada horizontal de manera efectiva. MySQL puede escalar en ciertos escenarios, pero con limitaciones en consultas complejas. SQLite está pensado para cargas pequeñas, mientras que SQL Server destaca en entornos empresariales, aunque su configuración para escalabilidad horizontal puede ser compleja y costosa.
- Facilidad de Uso:
  MySQL y SQLite son conocidos por su simplicidad en la configuración inicial. PostgreSQL, aunque requiere una configuración cuidadosa para optimizar su rendimiento, cuenta con una documentación extensa. SQL Server puede ser más complejo y costoso de administrar, y su dependencia del ecosistema Microsoft añade otra capa de complejidad.
- Seguridad:
  Tanto PostgreSQL como SQL Server ofrecen robustas medidas de seguridad nativas. MySQL proporciona seguridad adecuada para muchas aplicaciones, mientras que SQLite, por ser un motor embebido, puede no ser la opción ideal para aplicaciones con alta concurrencia y necesidades avanzadas de seguridad.
- Casos de Uso:
  PostgreSQL es ideal para aplicaciones que requieren una alta integridad de datos, flexibilidad en consultas avanzadas y un manejo eficiente de datos complejos. MySQL es excelente para proyectos web de transacciones rápidas y fáciles de administrar, y SQLite es preferible para prototipos o aplicaciones de baja demanda. SQL Server es adecuado para sistemas empresariales críticos donde el costo no es una limitación.

Comparación de Tecnologías

Persistencia Datos

POSTGRESQL	MYSQL	VS	SQL SERVER	SQLITE
Relacional	Relacional	TIPO	Relacional	Relacional Embebida
Open Source	Open Source y Comercial	LICENCIA	Comercial	Open Source
Excelente	Buena	INTEGRIDAD REFERENCIAL	Excelente	Básica
Vertical: Alta Horizontal: Moderada	Vertical: Buena Horizontal: Baja	ESCALABILIDAD	Vertical: Alta Horizontal: Moderada	Vertical: Limitada Horizontal: Limitada
Alta	Moderada	COMPLEJIDAD DATOS	Alta	Baja
Excelente	Bueno	RENDIMIENTO CONSULTAS	Excelente	Moderado
• Linux • Windows • MacOS	Principalmente • Windows • MacOS	COMPATIBILIDAD MULTIPLATAFORMA	Principalmente • Windows	• Linux • Windows • MacOS
Excelente	Excelente	COMUNIDAD Y DOCUMENTACION	Buena	Excelente

Justificación de la Elección

Tras evaluar las ventajas y desventajas de cada tecnología en función de los criterios establecidos, se opta por PostgreSQL por las siguientes razones:

- Integridad y Flexibilidad en el Manejo de Datos: PostgreSQL destaca por su cumplimiento robusto de transacciones ACID, lo que garantiza una alta integridad de los datos. Su excelente soporte para tipos de datos complejos (como JSON y arreglos) permite manejar de forma eficiente la diversidad de información que requiere la plataforma.
- Rendimiento y Optimización: Gracias al uso de índices avanzados y optimización para consultas complejas, PostgreSQL ofrece un rendimiento sobresaliente en escenarios donde se requiere procesamiento intensivo de datos. Esto resulta fundamental para una plataforma que necesita manejar múltiples usuarios y consultas avanzadas sin comprometer la velocidad y la eficiencia.
- Escalabilidad y Adaptabilidad: La capacidad de PostgreSQL para escalar verticalmente y de forma moderada horizontal permite adaptarse al crecimiento del sistema, asegurando que la base de datos pueda evolucionar conforme aumente la carga y la complejidad de las consultas. Si bien SQL Server ofrece una escalabilidad excelente, los costos y la complejidad de su configuración no se ajustan tan bien a los requerimientos de una solución escalable y accesible.

- Facilidad de Uso y Comunidad de Soporte: Aunque PostgreSQL requiere una configuración cuidadosa, la abundante documentación y el soporte de una extensa comunidad facilitan la resolución de problemas y la optimización de la base de datos.
  Esto permite que el equipo de desarrollo se enfoque en mejorar la plataforma sin enfrentar barreras significativas de aprendizaje.
- Seguridad y Confiabilidad:
  Las medidas de seguridad integradas en PostgreSQL aseguran que la información sensible esté protegida, lo cual es esencial en una plataforma que maneja datos críticos y transaccionales. Esta característica, combinada con su robustez y fiabilidad, lo posiciona como la opción ideal frente a alternativas que pueden requerir configuraciones adicionales o depender de entornos comerciales.

Referencias

Google Developers. (2023). *Angular documentation*. https://angular.io/docs

Meta Open Source. (2023). *React – A JavaScript library for building user interfaces*. https://react.dev/

Vercel. (2023). *Next.js Documentation*. https://nextjs.org/docs

You, E. (2023). *Vue.js – The Progressive JavaScript Framework*. https://vuejs.org/

Hartl, M. (2022). *Ruby on Rails Tutorial: Learn Web Development with Rails* (7th ed.). Addison-Wesley.

Holovaty, A., & Kaplan-Moss, J. (2023). *The Django Book*. https://djangobook.com

Lutz, M. (2021). *Learning Python* (5th ed.). O'Reilly Media.

Otwell, T. (2023). *Laravel documentation*. https://laravel.com/docs

Walls, C. (2022). *Spring Boot in Action* (2nd ed.). Manning Publications.

Momjian, B. (2001). PostgreSQL: introduction and concepts (Vol. 192, p. 2001). New York: Addison-Wesley.

DuBois, P. (2013). MySQL. Addison-Wesley.

Anexos

Anexo 1. Desglose de la planificación grupal

Tarea	Encargado	Fecha	Inicio	Fin
Asignación de tareas y organización concerniente a la entrega	André Pivaral y Roberto Nájera	17/03/2025	23:00	00:00
Elaborar Requisitos funcionales, asociando historias de usuario	Pablo Méndez	18/03/2025	08:00	10:00
Realizar diagramas de clases, de paquetes y de clases persistentes, y sus descripciones	Roberto Nájera	18/03/2025	08:00	10:00
Diagrama de Entidad Relación	André Pivaral	18/03/2025	08:00	10:00
Comparaciones de Frameworks para Frontend, Backend y DBMS para la presentación	André Pivaral, Luis Palacios, Pablo Méndez	19/03/2025	10:00	11:00
Selección de las tecnologías a usar basadas en los requisitos con sus descripciones detalladas	Luis Palacios	19/03/2025	11:00	12:00
Selección del DBMS	André Pivaral	19/03/2025	11:00	12:00
Diseño de la presentación a realizar y correcciones generales al documento	André Pivaral, Roberto Nájera Luis Palacios	19/03/2025	12:00	14:00
Gestión del repositorio y	Roberto Nájera	19/03/2025	22:00	0:00

informe de gestión grupal				

Enlace dentro del repositorio para cada informe LOGT: https://github.com/Ultimate-Truth-Seeker/ProyectoIS/tree/main/Corte3/informes

Anexo 2. Informe de gestión grupal del tiempo:

Informe de Gestión del Tiempo del Equipo: Corte 3
Fecha de Inicio de Trabajo: 17/03/2025 a las 18:00
Fecha de Finalización de Trabajo: 19/03/2025 a las 23:59

Puntos positivos
Hubo un buen manerjo de la comunicación, hubo claridad entre las diferentes partes
Todos completaron y entregaron sus tareas asignadas para esta entrega
Hubo flexibilidad y buen uso de las herramientas de trabajo

Puntos a mejorar
Mejorar la disponibilidad y manejo de tiempo para no trabajar al margen
Evitar procrastinar actividades y significativas para el avance del proyecto
Mantener con más esmero la puntualidad y el orden de los tiempos de entrega