Plan Final Proyecto Integrador

Equipo:

David Mauricio Chay Marín

Yahir Alberto Ordoñez Puc

Ángel Manuel Canché Aguayo

Universidad Tecnológica Metropolitana

Evaluación y Mejora para el Desarrollo de Software

Mtra. Marlene Ruiz Barboza

4º Cuatrimestre Grupo F

Parcial 3

26 de noviembre de 2024

Contenido

Introducción

El presente documento tiene como objetivo principal presentar nuestra plataforma, diseñada para simplificar significativamente la gestión de asistentes en diversos contextos, tales como convenciones, eventos, pláticas y paneles que se realizan anualmente. Uno de los problemas más comunes en este tipo de actividades es la formación de largas y tediosas colas de espera, especialmente al inicio de los eventos. Estas filas suelen generarse debido a que muchas organizaciones no priorizan adecuadamente los procesos de registro y validación de boletos, enfocándose únicamente en la experiencia final de los asistentes. Sin embargo, este descuido puede tener consecuencias negativas graves, ya que las largas esperas afectan la reputación del evento, disminuyen la satisfacción de los asistentes y, en algunos casos, desincentivan a potenciales participantes a asistir en futuras ediciones.

En este contexto, surge la necesidad imperiosa de desarrollar un sistema integral que aborde las principales deficiencias en la gestión de eventos. Nuestra propuesta consiste en una plataforma tecnológica que permita centralizar y optimizar todos los procesos relacionados con el registro, la gestión, la validación y el análisis de datos de los asistentes. Esto incluye funcionalidades como la creación, venta, compra y administración de boletos, definición de categorías, registro de usuarios, asociación de usuarios a eventos específicos, validación automatizada de boletos mediante herramientas como códigos QR, y generación de análisis detallados sobre la asistencia y el desempeño del evento.

Actualmente, muchos de los eventos de menor escala carecen de sistemas tecnológicos que les permitan consolidar estas tareas en un solo lugar. Esto genera errores frecuentes, desde problemas operativos que derivan en filas interminables hasta experiencias negativas que afectan tanto a los asistentes como a los organizadores. Nuestra plataforma busca solucionar esta brecha tecnológica proporcionando una herramienta confiable, eficiente y sencilla, capaz de garantizar una experiencia fluida y satisfactoria para todos los involucrados.

I. Requerimientos

El software RegCon tiene como propósito resolver problemas clave asociados con la gestión de registros y validación en convenciones y eventos. El sistema centralizará todos los procesos relacionados con:

- Creación de boletos: Permite definir categorías específicas como General y VIP, ajustando precios y beneficios según las necesidades del evento, ofreciendo flexibilidad para adaptarse a distintos tipos de convenciones.
- Gestión de asistentes: Proporciona herramientas para el registro, validación y administración de datos, asegurando un control eficiente y preciso de los participantes durante todo el evento.
- Validación de boletos: Implementa un sistema de códigos QR, diseñado para garantizar un acceso rápido y seguro. Incluye funcionalidades offline, ideales para escenarios donde las conexiones a Internet son débiles o intermitentes.
- Análisis e informes: Genera reportes detallados sobre métricas clave como asistencia y ventas de boletos, proporcionando datos relevantes que apoyan la toma de decisiones estratégicas por parte de los organizadores.
- Plataforma de boletos: Funciona como un complemento que facilita la venta y compra de entradas, centralizando este proceso y mejorando la experiencia del usuario final.

Con el objetivo de reducir filas de registro de hasta 19 horas a menos de 4 horas, RegCon también integrará herramientas de automatización respaldadas por inteligencia artificial y una interfaz amigable, diseñada para garantizar que organizadores y asistentes puedan utilizarla sin complicaciones. Este enfoque busca mejorar tanto la eficiencia operativa como la experiencia de los usuarios, posicionándose como una solución innovadora en la gestión de eventos.

II. I. Recopilación de Requerimientos

La recopilación de requerimientos para el desarrollo de RegCon se realizó a través de un enfoque estructurado y multicanal, que permitió identificar las necesidades y problemas más críticos en la gestión de eventos. Las principales actividades realizadas fueron:

II. II. I. Entrevistas a Organizadores

Se llevaron a cabo entrevistas con organizadores y miembros del staff de diversos eventos para entender las limitaciones y desafíos desde una perspectiva administrativa. Los principales hallazgos incluyeron:

- Limitaciones actuales:
    - Uso de herramientas básicas y no integradas como Excel, RegFox, y otros sistemas gratuitos, los cuales no están diseñados específicamente para la gestión de eventos.
    - Preferencia por el registro manual en libretas, especialmente en eventos de menor presupuesto, debido al alto costo percibido de desarrollar un software personalizado.
- Problemas recurrentes:
    - Falta de integración en un solo sistema que gestione boletos, registros, y asistencia.
    - Ausencia de herramientas tecnológicas robustas que puedan manejar grandes volúmenes de asistentes sin fallos.
    - Reticencia de algunos organizadores a priorizar la etapa de registro, considerándola de menor importancia, lo que genera deficiencias críticas en la experiencia de los asistentes.

II. II. II. Encuestas a Asistentes

Se realizaron encuestas a más de 1,200 asistentes de distintos eventos para identificar el impacto directo de los problemas de gestión en su experiencia. Los resultados destacaron:

- Tiempos de espera excesivos:
    - El 85% de los asistentes reportó tiempos de espera de 2 a más de 12 horas, lo que representa un gran desafío en términos de eficiencia operativa.
- Impacto psicológico:
    - El 58% de los encuestados mencionó haber experimentado aburrimiento mientras hacían fila, el 23% reportó estrés o ira, y el 19% dijo sentir ansiedad.
- Restricciones durante la espera:
    - El 68% de los asistentes evitó ir al baño por miedo a perder su lugar en la fila.
    - El 27% reportó no comer durante la espera, lo que refleja un impacto en su bienestar físico.
    - El 5% soportó dolores musculares por no abandonar su lugar, lo que indica un gran nivel de incomodidad.

Plan Final

II. II. III. Estudio de Casos

Se analizaron eventos específicos para evaluar cómo los problemas de gestión afectan la reputación y operatividad de los mismos. Entre los casos destacados están:

- Confuror 2024: Se evidenció un colapso del sistema de registro basado en Telegram, lo que provocó filas de hasta 19 horas en espacios reducidos, afectando tanto la experiencia de los asistentes como la reputación del evento.
- Tsunami 2019: El registro manual en libretas generó confusión, errores y tiempos de espera prolongados, resultando en una experiencia negativa para los asistentes.
- Otros eventos como Expo Terror también enfrentaron problemas similares, lo que demuestra que las fallas en el registro son un problema generalizado.

II. II. IV. Análisis de Comentarios

Se recopilaron opiniones y comentarios de asistentes y organizadores sobre sus experiencias en eventos anteriores, los cuales revelaron frustraciones significativas. Algunas frases recurrentes fueron:

- "Mala experiencia."
- "No asistiré de nuevo."
- "Esperé horas para nada." Estas respuestas reflejan un impacto directo en la percepción de los eventos y la fidelidad de los asistentes.

Adicionalmente, el análisis destacó que:

- El 73% de los encuestados estaría dispuesto a pagar una licencia única por una solución tecnológica como RegCon.
- Un 16% preferiría pagar una cuota mensual, mientras que solo un 3% manifestó no estar dispuesto a invertir en una solución.

Estos hallazgos subrayan la importancia de desarrollar un sistema que no solo resuelva los problemas actuales, sino que también sea accesible económicamente para los organizadores y atractivo para los usuarios finales.

I. III. Especificación de Requerimientos

I. III. I. Requerimientos Funcionales

1. Gestión de Eventos (RF-001):
    - Crear Eventos: Permitir a los organizadores crear nuevos eventos, especificando detalles como nombre, fecha, lugar, y capacidad máxima de asistentes.
    - Modificar Eventos: Posibilidad de editar información de eventos existentes.
    - Eliminar Eventos: Opción para eliminar eventos que ya no son necesarios.
    - Visualizar Eventos: Mostrar una lista de eventos programados, incluyendo estado (activo, completado, cancelado).
    - Prioridad: Obligatorio
2. Gestión de Boletos (RF-002):
    - Crear Boletos: Permitir la creación de diferentes tipos de boletos (general, VIP, etc.) con precios y beneficios específicos.
    - Modificar Boletos: Opción para editar detalles de boletos existentes.
    - Eliminar Boletos: Capacidad de eliminar boletos que no se venderán.
    - Visualizar Boletos: Mostrar información sobre los boletos disponibles, incluyendo cantidad restante.
    - Prioridad: Obligatorio
3. Registro de Asistentes (RF-003):
    - Registro de Usuarios: Permitir a los usuarios registrarse en el sistema, proporcionando información básica (nombre, correo, etc.).

- Validación de Boletos: Implementar un sistema para validar boletos utilizando códigos QR o identificadores únicos.
- Registro en el Evento: Proporcionar una interfaz para registrar a los asistentes al llegar al evento, incluyendo escaneo de códigos QR para agilizar el proceso.
- Prioridad: Obligatorio

4. Gestión de Asistentes (RF-004):
    - Visualización de Asistentes: Mostrar la lista de asistentes registrados para cada evento, con la opción de filtrar por diferentes criterios.
    - Modificar Información del Asistente: Permitir cambios en la información del asistente (por ejemplo, actualización de datos personales).
    - Registro a Actividades: Facilitar el registro de asistentes a diferentes actividades o paneles dentro del evento.
    - Prioridad: Obligatorio
5. Análisis e Informes (RF-005):
    - Generación de Informes: Crear informes imprimibles sobre asistencia, venta de boletos, y otras métricas relevantes.
    - Análisis de Datos: Proporcionar herramientas para analizar la información de los asistentes, identificando tendencias y patrones.
    - Diseño Intuitivo: Crear una interfaz fácil de usar que permita a los organizadores gestionar eventos y registros sin complicaciones.
    - Soporte Multiplataforma: Asegurar que el software sea accesible en dispositivos móviles y de escritorio.
    - Prioridad: Recomendable
6. Integración de QR y Validación (RF-006):

- Generación de Códigos QR: Implementar la generación automática de códigos QR al crear boletos.
- Escaneo de QR: Proporcionar herramientas para escanear códigos QR en la entrada y validar la autenticidad del boleto. 8.- Funcionalidades Offline
- Modo Offline: Permitir el funcionamiento del software sin conexión a Internet, registrando información localmente y sincronizándola una vez que se restablezca la conexión.
- Prioridad: Obligatorio

7. Soporte y Mantenimiento (RF-007):
    - Documentación: Proporcionar documentación y tutoriales para guiar a los usuarios en el uso del software.
    - Soporte Técnico: Implementar un sistema de soporte técnico para ayudar a los organizadores a resolver problemas o dudas.
    - Protección de Datos: Asegurar la protección de la información personal de los asistentes y organizadores.
    - Control de Acceso: Implementar diferentes niveles de acceso para organizadores, personal y asistentes.
    - Prioridad: Obligatorio
8. Funciones Adicionales (RF-008):
    - Integración con Redes Sociales: Permitir que los usuarios compartan su registro o experiencia en redes sociales.
    - Herramientas de IA: Incorporar funcionalidades que usen inteligencia artificial para prever la asistencia y optimizar el registro.
    - Prioridad: Recomendable

I. III. II. Requerimientos No Funcionales

1. Usabilidad (RN-001):
    - Interfaz Intuitiva: El sistema debe tener una interfaz de usuario que sea fácil de entender y navegar, permitiendo a los organizadores y asistentes realizar tareas sin complicaciones.
    - Documentación de Usuario: Proporcionar documentación clara y accesible que explique cómo usar el software, incluyendo tutoriales y guías.
    - Prioridad: Obligatorio
2. Desempeño (RN-002):
    - Tiempo de Respuesta: El sistema debe tener un tiempo de respuesta bajo (menos de 2 segundos) para las operaciones críticas, como la validación de boletos y el registro de asistentes.
    - Escalabilidad: El software debe ser capaz de manejar un aumento en la carga de usuarios sin degradar su desempeño, especialmente durante los picos de tráfico en eventos.
    - Prioridad: Obligatorio
3. Seguridad (RN-003):
    - Protección de Datos: Implementar cifrado para la transmisión de datos sensibles, como información personal y detalles de pago.
    - Autenticación y Autorización: El sistema debe contar con mecanismos robustos de autenticación y autorización para proteger el acceso a la información sensible.
    - Respaldo de Datos: Realizar copias de seguridad regulares de la base de datos para evitar la pérdida de información crítica.
    - Prioridad: Obligatorio
4. Compatibilidad (RN-004):

- Multiplataforma: El software debe ser compatible con diferentes dispositivos (PC, tabletas, smartphones) y sistemas operativos (Windows, macOS, Android, iOS).
- Integración con Otras Herramientas: Debe permitir la integración con otros sistemas y plataformas, como herramientas de pago, servicios de correo electrónico, y aplicaciones de redes sociales.
- Prioridad: Recomendable

5. Mantenibilidad (RN-005):
    - Código Limpio y Documentado: El código fuente debe ser claro y estar bien documentado para facilitar futuras actualizaciones y mantenimiento.
    - Modularidad: El software debe estar estructurado de manera modular, permitiendo que componentes individuales sean actualizados o reemplazados sin afectar el sistema en su totalidad.
    - Prioridad: Obligatorio
6. Disponibilidad (RN-006):
    - Alta Disponibilidad: El sistema debe estar disponible para los usuarios en todo momento, especialmente durante los eventos, con un tiempo de inactividad mínimo.
    - Funcionalidad Offline: Debe ofrecer funcionalidades esenciales incluso sin conexión a Internet, permitiendo el registro y validación de boletos de manera local.
    - Prioridad: Obligatorio
7. Rendimiento del Sistema (RN-007):
    - Capacidad de Manejar Cargas Elevadas: Debe ser capaz de gestionar un gran número de registros simultáneos, especialmente durante picos de tráfico, como el inicio de eventos.
    - Optimización de Recursos: El software debe utilizar eficientemente los recursos del sistema (CPU, memoria) para evitar problemas de rendimiento.

- Accesibilidad: Cumplimiento con Estándares de Accesibilidad: El software debe cumplir con las pautas de accesibilidad (como WCAG) para asegurar que sea usable por personas con discapacidades.
- Prioridad: Obligatorio

8. Portabilidad (RN-008):
    - Despliegue en Diferentes Entornos: El software debe poder ser instalado y ejecutado en diferentes entornos de hardware y software sin problemas.
    - Prioridad: Obligatorio
9. Interoperabilidad (RN-009):
    - Comunicación entre Sistemas: Debe ser capaz de interactuar con otros sistemas y aplicaciones, facilitando la transferencia de datos y la colaboración entre plataformas.
    - Prioridad: Recomendable

I. IV. Validación de Requerimientos

I. IV. I. Prototipos

- Descripción:
    - Los prototipos representan versiones preliminares y funcionales de las principales interfaces y módulos del sistema RegCon. Estos prototipos serán diseñados para facilitar pruebas tempranas y recopilar retroalimentación antes del desarrollo completo.
    - Los prototipos incluirán:
        - Interfaz de Registro de Asistentes: Formulario simplificado que permita registrar la información básica de un usuario (nombre, correo, tipo de boleto) y asociarlo con un evento.

- Validación de Boletos: Escáner QR interactivo que simule la validación de un boleto, destacando errores comunes (por ejemplo, boleto inválido o ya utilizado).
- Panel de Gestión de Eventos para Organizadores: Incluye opciones para crear, modificar y eliminar eventos, gestionar la capacidad y definir categorías de boletos.
- Generación de Reportes: Prototipo básico que permita generar y visualizar un informe de asistencia, mostrando datos clave como total de asistentes y tiempos promedio de validación.
- Los prototipos también incluirán opciones de accesibilidad y usabilidad para verificar que cumplan con estándares de diseño intuitivo.

- Objetivo:
  - Validar el diseño de las interfaces y flujos principales del sistema.
  - Asegurar que las funcionalidades sean comprensibles y accesibles para organizadores y asistentes.
  - Identificar áreas de mejora a través de pruebas con usuarios finales antes del desarrollo completo.
- Actividades:
  - Pruebas de Navegación: Los usuarios interactuarán con las interfaces simulando tareas reales, como registrar asistentes o validar boletos.
  - Evaluación de Usabilidad: Se medirá la facilidad de uso, claridad de los menús y tiempo promedio para completar acciones clave.
  - Iteración y Ajustes: La retroalimentación obtenida será utilizada para realizar mejoras en los prototipos.

Plan Final

I. IV. II. Casos de Uso

Los casos de uso describen escenarios concretos donde los usuarios interactúan con el sistema. A continuación, se presentan varios ejemplos:

- Caso 1: Registro de Asistentes
    - Actor: Staff de registro.
    - Descripción: Un miembro del staff registra manualmente a un asistente que llega al evento y no realizó preregistro en línea.
    - Flujo:
        - El staff ingresa el nombre, correo y número de boleto del asistente en el sistema.
        - El sistema verifica la validez del boleto.
        - El asistente es registrado exitosamente y su información se almacena.
- Caso 2: Validación de Boletos en la Entrada
    - Actor: Personal de validación.
    - Descripción: Un asistente llega a la entrada del evento con su boleto en formato digital.
    - Flujo:
        - El personal escanea el código QR del boleto utilizando la aplicación.
        - El sistema verifica la validez del boleto en menos de 2 segundos.
        - Si es válido, el sistema registra la entrada y muestra un mensaje de confirmación.
- Caso 3: Creación de un Evento
    - Actor: Organizador.

    - Descripción: Un organizador crea un nuevo evento para una convención en la plataforma.
    - Flujo:
        - El organizador ingresa los datos del evento (nombre, fecha, lugar, capacidad máxima).
        - Define las categorías de boletos (General, VIP) y sus precios.
        - El sistema guarda el evento y genera automáticamente un espacio de trabajo asociado.

- Caso 4: Generación de Reportes
    - Actor: Organizador.
    - Descripción: Tras finalizar un evento, el organizador genera un reporte detallado de asistencia.
    - Flujo:
        - Selecciona el evento desde el panel de administración.
        - Elige las métricas que desea incluir (número total de asistentes, tiempo promedio de espera, ventas por categoría de boletos).
        - El sistema genera un documento descargable con gráficos y estadísticas clave.
- Caso 5: Funcionalidades Offline
    - Actor: Staff en áreas con conectividad limitada.
    - Descripción: Un asistente llega con su boleto y el staff utiliza la funcionalidad offline para validar su entrada.
    - Flujo:
        - El staff escanea el código QR del boleto.
        - El sistema registra la validación localmente.

- Una vez restaurada la conexión, los datos se sincronizan automáticamente.

I. IV. III. Historias de Usuario

Las historias de usuario permiten entender las necesidades desde la perspectiva de los diferentes roles del sistema:

- Organizador:
  - "Como organizador, quiero poder crear un evento con diferentes categorías de boletos, para atender las necesidades de mis asistentes."
  - "Como organizador, quiero generar reportes de asistencia al final del evento, para evaluar su éxito y planificar futuras mejoras."
- Asistente:
  - "Como asistente, quiero recibir un boleto con un código QR para no tener que imprimirlo y facilitar mi ingreso."
  - "Como asistente, quiero validar mi boleto rápidamente para evitar filas largas y ahorrar tiempo."
- Staff de Registro:
  - "Como miembro del staff, quiero registrar asistentes manualmente cuando no tienen pre-registro, para que puedan acceder al evento."
  - "Como staff, quiero que el sistema sea fácil de usar y valide boletos en menos de 2 segundos para evitar retrasos."
- Personal Técnico:
  - "Como personal técnico, quiero que el sistema registre los datos incluso sin conexión a Internet para garantizar su uso en todo momento."
  - "Como personal técnico, quiero poder recuperar datos en caso de fallos para evitar pérdidas de información.

Plan Final

I. IV. IV. Listado de Requerimientos / Backlog

ID	Descripción del Requerimiento/Tarea	Tipo	Prioridad	Estado	Responsable	Notas
1	Crear sistema para registro de asistentes.	Funcional	Alta	Pendiente	David Chay	Incluir validación de campos obligatorios.
2	Implementar validación de boletos mediante QR.	Funcional	Alta	Pendiente	David Chay	Asegurar funcionalidad offline.
3	Desarrollar módulo de gestión de eventos (crear, modificar, eliminar).	Funcional	Alta	Pendiente	David Chay	Debe soportar hasta 500 eventos activos.
4	Incorporar generación automática de reportes de asistencia y ventas.	Funcional	Media	Pendiente	David Chay	Incluir gráficos exportables.
5	Diseñar interfaz intuitiva para organizadores y asistentes.	No Funcional	Alta	En Progreso	Yahir Ordoñez	Basarse en pruebas de prototipos.

6	Añadir funcionalidad de escaneo de QR para validación rápida (<2 segundos).	Funcional	Alta	Pendiente	David Chay	Compatibilidad con dispositivos móviles.
7	Implementar sistema de control de acceso por roles.	No Funcional	Alta	Pendiente	Yahir Ordoñez	Roles: administrador, staff, asistente.
8	Asegurar compatibilidad multiplataforma (Windows, macOS, Android, iOS).	No Funcional	Media	Pendiente	David Chay	Pruebas en diferentes sistemas operativos.
9	Configurar funcionalidad offline para validación y sincronización posterior.	Funcional	Alta	Pendiente	David Chay	Importante para eventos con conectividad limitada.
10	Crear módulo para generación de boletos personalizados.	Funcional	Alta	Pendiente	Yahir Ordoñez	Debe incluir precios, categorías (VIP, General).

11	Incorporar cifrado de datos personales y detalles de pago.	No Funcional	Alta	Pendiente	Yahir Ordoñez	Cumplir con estándares GDPR.
12	Implementar sistema de respaldo de datos automatizado.	No Funcional	Media	Pendiente	Ángel Canché	Configuración de backups semanales.
13	Añadir herramientas de análisis predictivo con inteligencia artificial.	Funcional	Baja	Pendiente	Ángel Canché	Utilizar datos históricos para predecir tendencias.
14	Diseñar sistema de integración con redes sociales.	Funcional	Baja	Pendiente	Yahir Ordoñez	Permitir compartir registro y boletos.
15	Crear documentación técnica y tutoriales para usuarios finales.	No Funcional	Media	Pendiente	Ángel Canché	Videos cortos y manuales en PDF.

I. V. Control de Cambios en los Requerimientos

El control de cambios en los requerimientos es un proceso esencial para mantener la alineación entre los objetivos de nuestro proyecto y las modificaciones que surgen a lo largo de su desarrollo. Este proceso se enfoca en garantizar que todos los cambios sean documentados, evaluados y aprobados de manera estructurada, minimizando riesgos y asegurando que el sistema final cumpla con las expectativas de los usuarios y stakeholders.

I. V. I. Registro de Cambios

Cada cambio propuesto será registrado formalmente en un sistema dedicado que permita el seguimiento detallado de las solicitudes. El registro debe incluir:

- Identificador único del cambio.
- Fecha de la solicitud.
- Descripción detallada del cambio.
- Origen del cambio: Quién lo solicita (stakeholders, equipo técnico, usuarios finales).
- Estado del cambio: Pendiente, en evaluación, aprobado, implementado o rechazado.

I. V. II. Evaluación del Impacto

Los cambios serán evaluados para determinar su viabilidad técnica, impacto en el cronograma y presupuesto, así como su alineación con los objetivos del proyecto. La evaluación incluirá:

- Viabilidad técnica: Análisis de la factibilidad de implementar el cambio con los recursos disponibles.
- Impacto en el cronograma: Identificación de posibles retrasos o ajustes necesarios.
- Costo asociado: Estimación de recursos adicionales que puedan requerirse.
- Prioridad: Clasificación del cambio como alto, medio o bajo impacto según su criticidad.

I. V. III. Aprobación de Cambios

Solo los cambios aprobados mediante un proceso formal serán implementados. La aprobación estará a cargo de un comité o grupo designado, que considerará los resultados de la evaluación de impacto y los objetivos del proyecto.

I. V. IV. Implementación y Validación

Una vez aprobado, el cambio será:

1. Asignado a un responsable: El encargado de implementar el cambio lo registrará en el sistema de control de versiones.
2. Validado: Se realizarán pruebas para asegurar que el cambio cumple con lo solicitado y no afecta negativamente otras funcionalidades del sistema.

I. V. V. Actualización de Documentación

Después de la implementación, toda la documentación relevante será actualizada para reflejar los cambios realizados, asegurando consistencia y trazabilidad en el proyecto.

I. V. VI. Herramientas para el Control de Cambios

- Gestión de Solicitudes: Herramientas como Jira, se utilizarán para registrar, rastrear y priorizar solicitudes de cambio.
- Herramienta Supabase: Supabase será utilizada como una herramienta centralizada para gestionar el flujo de datos y los registros clave del proyecto. Esta plataforma permitirá un control eficiente de la información almacenada, asegurando una autenticación robusta y confiable para todos los usuarios involucrados en el sistema.
- Control de Versiones: Uso de sistemas como Git para mantener un historial detallado de modificaciones en el código fuente.
- Documentación Centralizada: Plataformas como Confluence se emplearán para documentar cada cambio y su impacto.

Plan Final

I. V. VII. Criterios de Evaluación para Cambios

- Urgencia: ¿Es el cambio crítico para el funcionamiento del sistema?
- Beneficio: ¿Qué valor aporta el cambio al proyecto o al usuario final?
- Riesgo: ¿Qué tan probable es que el cambio introduzca complicaciones técnicas o de diseño?
- Costo: ¿El cambio es viable dentro del presupuesto asignado?

Este proceso asegura que los cambios en los requerimientos sean gestionados de manera estructurada, eficiente y transparente, contribuyendo al éxito del proyecto.

II. Diseño del Software

II. I. Selección del Diseño Arquitectónico

La selección del diseño arquitectónico para RegCon se realizó considerando las necesidades clave del sistema: escalabilidad, modularidad, mantenibilidad, seguridad y una experiencia de usuario fluida. Dado que el sistema está orientado a gestionar múltiples convenciones y eventos de manera independiente para diferentes administradores, se optó por una arquitectura basada en los principios de cliente-servidor y el uso de tecnologías modernas orientadas a la nube.

- Modelo Cliente-Servidor

El modelo arquitectónico elegido sigue un enfoque de cliente-servidor, donde:

  - El cliente es una aplicación desarrollada en React que funciona como la interfaz principal para los usuarios (administradores y asistentes). Este cliente consume datos y funcionalidades a través de una API RESTful.
  - El servidor, construido con Node.js y Express, actúa como el núcleo del sistema, gestionando la lógica empresarial y sirviendo como intermediario entre la base de datos y la interfaz de usuario.
  - Este diseño asegura una separación clara entre la capa de presentación (frontend) y la lógica de negocio (backend), lo que facilita la evolución y el mantenimiento del sistema.

- Arquitectura de Microservicios Modular

Aunque el sistema actualmente opera como un monolito modular, se consideró la posibilidad de evolucionar hacia una arquitectura de microservicios en futuras iteraciones para mejorar la escalabilidad. La modularización del backend asegura que las funcionalidades principales (gestión de usuarios, eventos, tickets, y asistencia) estén separadas en módulos bien definidos,

facilitando el trabajo en equipo y la posibilidad de agregar nuevas características sin afectar las existentes.

- Base de Datos Relacional en la Nube

La base de datos juega un papel crítico en el diseño arquitectónico. Se optó por una solución relacional basada en PostgreSQL alojada en la nube debido a:

  - Su capacidad para manejar relaciones complejas entre entidades como usuarios, eventos, tickets y grupos de trabajo.
  - Su soporte para escalabilidad horizontal y vertical.
  - Integración nativa con herramientas de gestión y análisis.
  - Cada administrador tiene acceso únicamente a los datos de su grupo de trabajo mediante el uso de ID de grupos de trabajo, lo que garantiza la segmentación de datos y la seguridad.
- Servicios en la Nube

Para asegurar un despliegue flexible y de bajo costo, el sistema fue diseñado para operar en plataformas de nube como Back4App, aprovechando su soporte para contenedores mediante Docker. Esto permite empaquetar la aplicación con todas sus dependencias y garantizar su correcto funcionamiento en diferentes entornos.

- Protocolo y Seguridad
  - Autenticación y Autorización: Se implementó un sistema de autenticación basado en JSON Web Tokens (JWT) para validar y proteger las sesiones de los usuarios.
  - Protección de rutas: Las rutas sensibles del cliente y el servidor están protegidas y sólo accesibles para usuarios autenticados con privilegios específicos.

    - Conexión segura: Todo el tráfico entre el cliente y el servidor está previsto para usar HTTPS, asegurando la encriptación de datos en tránsito.

- Escalabilidad y Mantenimiento

El diseño arquitectónico está pensado para crecer junto con las necesidades del sistema:

    - Escalabilidad horizontal: El servidor y la base de datos pueden escalar para manejar más usuarios y eventos mediante balanceo de carga y clústeres en la nube.
    - Mantenimiento: La separación de responsabilidades entre cliente, servidor y base de datos permite que cada capa se actualice o mantenga de forma independiente.

II. II. Diseño Detallado

El diseño detallado de RegCon se centra en la descripción específica de los componentes del sistema, su interacción, y la manera en que se implementan para satisfacer los requerimientos funcionales y no funcionales del proyecto. A continuación, se describen los elementos clave del diseño detallado, organizados por capas y funcionalidades principales.

II. II. I. Capa de Presentación

La capa de presentación está implementada con React, una biblioteca de JavaScript ampliamente adoptada para construir interfaces de usuario interactivas. Esta capa actúa como el punto de interacción principal para los usuarios finales, proporcionando una experiencia fluida y responsiva.

Componentes principales:

- Landing Page: Una página de inicio con secciones como 'Inicio,' '¿Cómo funciona?' y 'Contacto,' diseñada para atraer nuevos usuarios y proporcionar información sobre el sistema.

Plan Final

- Dashboard del Administrador: Incluye vistas para la gestión de eventos, tickets, y asistentes. Diseñado con componentes reutilizables como tablas dinámicas, formularios modulares y gráficos.
- Panel de Usuario Final: Proporciona a los asistentes una forma de ver eventos disponibles, registrarse, y gestionar sus tickets.
- Protección de Rutas: Implementada con react-router-dom para garantizar que los usuarios no autenticados sean redirigidos al login.

Flujo de Trabajo:

1. El usuario inicia sesión.
2. Dependiendo de sus permisos (administrador o asistente), accede a un conjunto de componentes.
3. Las acciones del usuario, como registrar un evento o comprar un ticket, desencadenan solicitudes a la API REST.

Tecnologías usadas:

- React para el desarrollo del frontend.
- Axios para la comunicación con la API.
- Context API o Redux para la gestión del estado de la aplicación.

II. II. II. Capa de Lógica de Negocios

El backend está implementado con Node.js y Express, organizando el código en controladores, servicios y middleware para asegurar una estructura limpia y mantenible.

Componentes principales:

- Rutas (Endpoints): Cada funcionalidad del sistema está asociada a una ruta específica en la API REST. Por ejemplo:
    - /api/users: Gestión de usuarios.
    - /api/events: Creación y edición de eventos.
    - /api/tickets: Gestión de tickets y categorías.
    - /api/attendance: Registro y validación de asistencia.
- Controladores: Encargados de manejar las solicitudes entrantes y llamar a los servicios correspondientes. Ejemplo:
    - EventController.js: Procesa solicitudes para crear, editar o eliminar eventos.
- Servicios: Contienen la lógica de negocio y son responsables de interactuar con la base de datos. Ejemplo:
    - UserService.js: Valida credenciales, crea usuarios, y gestiona permisos.
- Middleware de Seguridad: Verifica tokens JWT, filtra datos según el grupo de trabajo del usuario y protege rutas.

Flujo de trabajo típico:

1. Una solicitud HTTP llega a la API.
2. El middleware verifica la autenticación y autorización.
3. El controlador correspondiente procesa la solicitud y delega la lógica al servicio.
4. El servicio interactúa con la base de datos y devuelve una respuesta al controlador.
5. El controlador envía la respuesta al cliente.

Estructura del backend:

src/

├── controllers/

├── services/

├── middleware/

├── routes/

├── config/

└── models/

II. II. III. Capa de Datos

La base de datos está diseñada con PostgreSQL para aprovechar su robustez y capacidad de manejar relaciones complejas.

Estructura de la base de datos:

- Tablas principales:
    - users: Almacena información de usuarios, roles y credenciales.
    - events: Define los eventos creados por los administradores.
    - tickets: Detalla los tipos y categorías de tickets.
    - attendance: Registra la asistencia y validación de entradas.
    - workgroups: Agrupa usuarios, eventos y tickets según el administrador.

Relaciones clave:

- Cada usuario pertenece a un grupo de trabajo (workgroup_id).
- Los eventos, tickets y registros de asistencia están vinculados a un workgroup_id.

- Los usuarios pueden estar asociados a múltiples roles, permitiendo una jerarquía flexible.

Optimización de la base de datos:

- Uso de índices para acelerar consultas en campos de uso frecuente, como user_id y workgroup_id.
- Normalización para evitar redundancias y mantener integridad referencial.
- Implementación de claves foráneas para garantizar la consistencia entre tablas.

II. II. IV. Módulos Específicos

a. Gestión de Usuarios:

- Registro y autenticación mediante tokens JWT.
- Roles definidos para administradores y asistentes.
- Función de restablecimiento de contraseña.

b. Gestión de Eventos:

- Formularios dinámicos para crear y editar eventos.
- Filtrado por grupos de trabajo.
- Asociación de categorías de tickets a cada evento.

c. Validación de Asistencia:

- Escaneo de códigos QR para validar tickets.
- Generación de reportes de asistencia.
- Envío de notificaciones en tiempo real.

d. Sistema de Pagos:

- Integración con proveedores de pagos como Stripe o PayPal.
- Creación de facturas y confirmaciones automáticas.

II. II. V. Comunicación entre Componentes

El sistema utiliza un enfoque asíncrono para manejar la comunicación:

- Solicitudes del cliente al servidor a través de API REST.
- Respuestas en formato JSON para estandarizar la transferencia de datos.
- WebSockets (en futuras iteraciones) para actualizaciones en tiempo real, como la validación instantánea de tickets.

II. II. VI. Seguridad y Confiabilidad

- Cifrado de contraseñas utilizando bcrypt.
- Middleware personalizado para validar roles y permisos.
- Respaldo de la base de datos en intervalos regulares.

II. III. Diseño de la Base de Datos

El diseño de la base de datos para el sistema RegCon fue cuidadosamente planificado para garantizar que cumpla con los requerimientos funcionales y no funcionales del sistema. Se centra en la organización de los datos para optimizar la gestión de usuarios, eventos, tickets y asistencia, utilizando un enfoque relacional para mantener integridad y escalabilidad. A continuación, se describen en detalle los artefactos creados, las herramientas utilizadas, las convenciones de nombrado aplicadas y las prácticas implementadas para garantizar un modelo de datos sostenible y comprensible.

II. III. I. Diagrama Entidad-Relación

El modelo Entidad-Relación (E-R) fue diseñado utilizando herramientas especializadas como DBeaver para capturar visualmente las entidades, atributos y relaciones entre tablas del sistema. Este diagrama no solo sirve como una referencia técnica para desarrolladores y administradores de bases de datos, sino también como una herramienta de comunicación entre equipos técnicos y no técnicos.

Entidades Principales:

1. Usuario (usuario)
    - Almacena información sobre los usuarios del sistema, incluyendo administradores y asistentes.
    - Relacionado con los grupos de trabajo para determinar su contexto.
2. Grupo de Trabajo (grupo_trabajo)
    - Define las áreas administrativas dentro de las cuales se organizan eventos y usuarios.
    - Relacionado con eventos, tickets y asistencia.

3. Evento (evento)
    - Representa los eventos creados por los administradores, incluyendo detalles como la fecha, ubicación y descripción.
4. Ticket (ticket)
    - Describe los boletos disponibles para cada evento, incluyendo su tipo, precio y categoría.
5. Asistencia (asistencia)
    - Registra la validación de la entrada de los usuarios a los eventos mediante códigos QR.

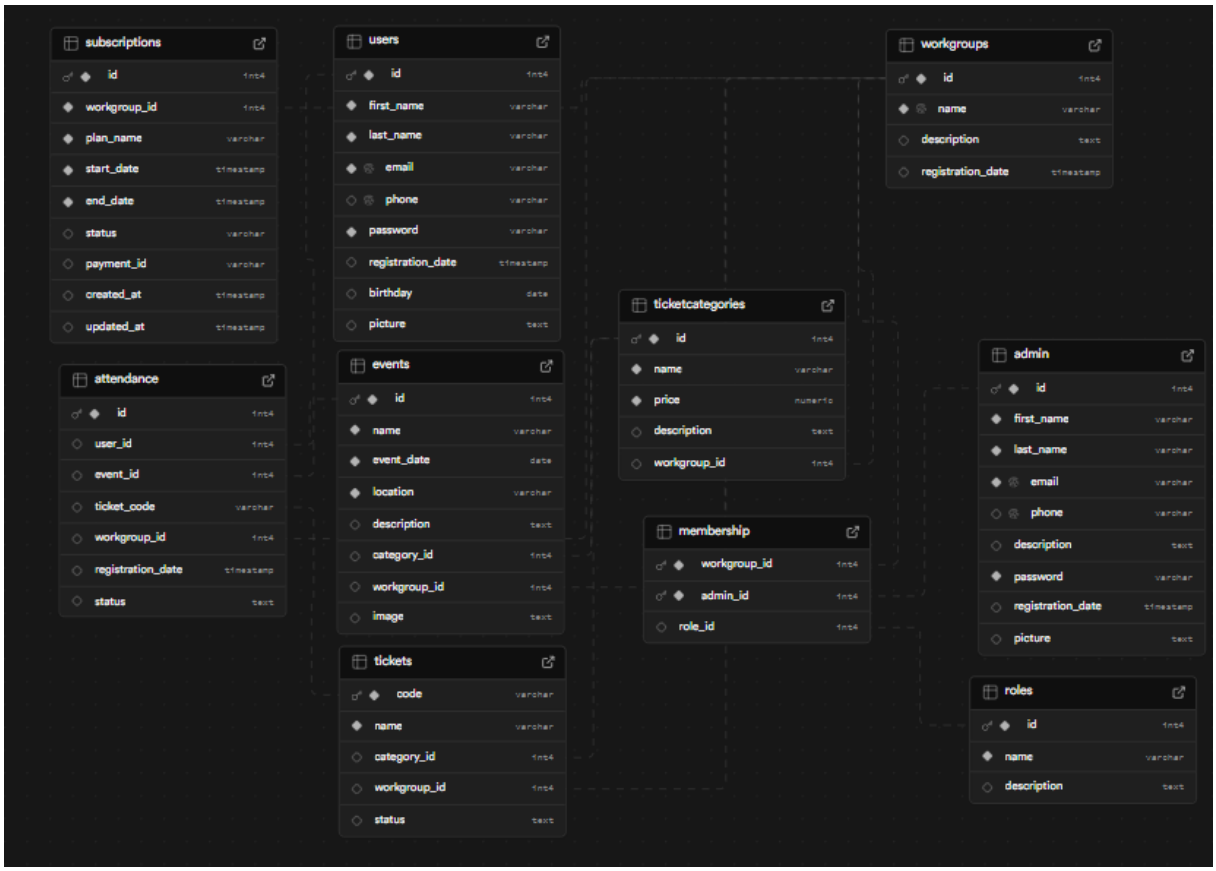
Relaciones Clave:

- Cada usuario pertenece a un único grupo de trabajo, pero un grupo de trabajo puede tener múltiples usuarios.
- Un evento está asociado a un único grupo de trabajo, pero un grupo de trabajo puede gestionar múltiples eventos.
- Los tickets están vinculados tanto a eventos como a usuarios.
- La asistencia es una relación transitiva entre usuarios, tickets y eventos.

El diagrama refleja estas relaciones utilizando llaves primarias y foráneas para garantizar la integridad referencial. Cada entidad incluye atributos relevantes y relaciones con cardinalidades bien definidas, por ejemplo:

- 1:N entre grupo de trabajo y eventos.
- N:M entre usuarios y eventos (a través de asistencia).

Plan Final


subscriptions
id int4
workgroup_id int4
plan_name varchar
start_date timestamp
end_date timestamp
status varchar
payment_id varchar
created_at timestamp
updated_at timestamp
users
id int4
first_name varchar
last_name varchar
email varchar
phone varchar
password varchar
registration_date timestamp
birthday date
picture text
workgroups
id int4
name varchar
description text
registration_date timestamp
attendance
id int4
user_id int4
event_id int4
ticket_code varchar
workgroup_id int4
registration_date timestamp
status text
events
id int4
name varchar
event_date date
location varchar
description text
category_id int4
workgroup_id int4
image text
ticketcategories
id int4
name varchar
price numeric
description text
workgroup_id int4
admin
id int4
first_name varchar
last_name varchar
email varchar
phone varchar
description text
password varchar
registration_date timestamp
picture text
membership
workgroup_id int4
admin_id int4
role_id int4
tickets
code varchar
name varchar
category_id int4
workgroup_id int4
status text
roles
id int4
name varchar
description text

Plan Final

II. III. II. Diccionario de Datos

El diccionario de datos documenta cada tabla, columna, y sus características, proporcionando una descripción clara del propósito de cada elemento en el modelo.

Tabla	Columna	Tipo de Dato	Descripción	Restricciones
usuario	id_usuario	serial	Identificador único del usuario	Llave primaria
	nombre	varchar(100)	Nombre del usuario	No nulo
	email	varchar(150)	Correo electrónico único	Único, no nulo
	password	varchar(255)	Contraseña cifrada del usuario	No nulo
	id_grupo_trabajo	integer	Referencia al grupo de trabajo del usuario	Llave foránea
grupo_trabajo	id_grupo_trabajo	serial	Identificador único del grupo	Llave primaria
	nombre_grupo	varchar(100)	Nombre del grupo de trabajo	No nulo
evento	id_evento	serial	Identificador único del evento	Llave primaria
	nombre_evento	varchar(100)	Nombre del evento	No nulo
	fecha_inicio	date	Fecha de inicio del evento	No nulo
	fecha_fin	date	Fecha de finalización del evento	No nulo
	id_grupo_trabajo	integer	Referencia al grupo de trabajo	Llave foránea

Nota: El diccionario se mantiene actualizado cada vez que se realizan cambios en la estructura de la base de datos, alineado con el principio de documentación continua.

II. III. III. Herramientas Utilizadas

Para la creación, documentación y mantenimiento del diseño de la base de datos, se utilizaron herramientas como:

- DBeaver: Para diseñar y mantener el diagrama E-R y visualizar relaciones complejas entre tablas.
- pgAdmin: Para gestionar directamente la base de datos PostgreSQL.

- Diagramas online (Lucidchart, Draw.io): Para compartir representaciones visuales del modelo con el equipo.

II. III. IV. Convenciones de Nombrado

Se adoptaron las siguientes convenciones de nombrado para estandarizar el modelo de datos y garantizar su claridad:

Nombrado de tablas:

1. Las tablas están en singular, en minúsculas y son pronunciables.
    - Ejemplo: usuario, evento, grupo_trabajo.
2. Las tablas transitivas se nombran comenzando con la entidad principal.
    - Ejemplo: usuario_evento, evento_ticket.

Nombrado de columnas:

1. Los nombres son descriptivos, en singular y en minúsculas.
    - Ejemplo: id_usuario, nombre_evento, fecha_inicio.
2. Las llaves primarias siguen el formato id_nombre_tabla.
    - Ejemplo: id_usuario, id_evento.
3. Las llaves foráneas utilizan el nombre de la columna referenciada.
    - Ejemplo: id_grupo_trabajo en la tabla usuario.

Estandarización adicional:

- No se usaron prefijos que indiquen el tipo de dato o el nombre de la tabla.
- Se evitaron abreviaturas no comunes o acrónimos.

II. III. V. Buenas Prácticas

- Normalización: Las tablas se diseñaron para minimizar redundancias y asegurar integridad referencial.
- Índices: Se añadieron índices en campos clave como email y id_evento para optimizar consultas frecuentes.
- Integridad referencial: Todas las relaciones están respaldadas por restricciones de claves foráneas.
- Copia de seguridad y versionado: Se realiza un respaldo regular de la base de datos, y los cambios estructurales son versionados.

II. IV. Versionado de la Base de Datos

El versionado de la base de datos de RegCon ha sido un aspecto fundamental del desarrollo del proyecto, permitiendo gestionar de manera organizada los cambios en la estructura del modelo de datos a lo largo de las iteraciones. Este enfoque ha garantizado que las actualizaciones sean controladas, documentadas y aplicadas de manera consistente en los entornos de desarrollo, prueba y producción. A continuación, se detalla el proceso de versionado implementado, el historial de cambios realizados y las herramientas utilizadas.

II. IV. I. Proceso de Versionado

El versionado de la base de datos sigue un enfoque estructurado que incluye las siguientes etapas:

1. Planeación de Cambios:
    - Antes de realizar cualquier cambio, los requerimientos se analizan para determinar cómo impactarán la estructura actual de la base de datos.

- Se generan propuestas de migración documentadas que describen las modificaciones planificadas (por ejemplo, agregar tablas, columnas, índices, o relaciones).

2. Creación de Migraciones:
    - Se crean scripts de migración utilizando herramientas especializadas como Knex.js o Sequelize (en caso de Node.js) para automatizar la aplicación de cambios.
    - Los scripts incluyen comandos SQL explícitos para añadir, modificar o eliminar elementos de la base de datos, garantizando la reversibilidad mediante scripts de "rollback."
3. Ejecución Controlada:
    - Los scripts de migración se aplican primero en entornos de desarrollo y prueba, asegurando que los cambios no introduzcan errores.
    - Una vez validados, se despliegan en producción utilizando herramientas de gestión de despliegue como Flyway o Liquibase.
4. Documentación:
    - Cada cambio se documenta con un historial de versiones, indicando:
        - Fecha y hora de la migración.
        - Autor de la modificación.
        - Descripción del cambio.
        - Justificación y requerimientos asociados.
5. Respaldo y Restauración:

- Antes de aplicar cualquier cambio, se realiza un respaldo completo de la base de datos.
- En caso de errores, los scripts de "rollback" permiten revertir los cambios aplicados.

II. IV. II. Herramientas Utilizadas

Para gestionar el versionado de la base de datos, se implementaron las siguientes herramientas:

- Knex.js: Utilizada como herramienta de migración en proyectos Node.js, permitiendo definir cambios en el esquema de la base de datos mediante un enfoque programático.
- Supabase: Para la visualización de estructuras de datos y generación manual de scripts SQL cuando era necesario. Además de ser el sitio donde se hosteó la base de datos desde el comienzo.
- pgAdmin: Herramienta complementaria para gestionar la base de datos PostgreSQL.
- Git: Para almacenar y versionar los scripts de migración junto con el código del backend, garantizando la sincronización entre el esquema de la base de datos y el código del sistema.

II. IV. III. Historial de Cambios

El proyecto RegCon ha experimentado múltiples iteraciones en el modelo de datos a medida que los requisitos evolucionaron. A continuación, se presenta un resumen detallado del historial de cambios más relevantes en la base de datos:

Versión 1.0: Estructura Inicial

- Fecha: 09 de octubre de 2024

Plan Final

- Cambios:
    - Creación de tablas iniciales: usuario, evento, ticket, y asistencia.
    - Definición de relaciones básicas entre usuarios y eventos (1:N).
    - Uso de claves primarias y foráneas para garantizar la integridad referencial.
    - Se definió un esquema inicial sin segmentación por grupos de trabajo.
- Motivación: Establecer un modelo básico para soportar las funcionalidades principales del sistema.

Versión 1.1: Introducción de Grupos de Trabajo

- Fecha: 12 de octubre de 2024
- Cambios:
    - Creación de la tabla grupo_trabajo para agrupar usuarios, eventos y tickets.
    - Adición de la columna id_grupo_trabajo en las tablas usuario, evento y ticket.
    - Actualización de relaciones para asociar registros con grupos de trabajo específicos.
- Motivación: Permitir que múltiples administradores gestionen sus datos de forma independiente.

Versión 1.2: Optimización y Normalización

- Fecha: 15 de octubre de 2024
- Cambios:
    - Eliminación de redundancias mediante normalización de tablas.

    - Introducción de índices en las columnas email y id_grupo_trabajo para optimizar consultas frecuentes.
    - Ajustes en tipos de datos para mejorar consistencia (e.g., varchar a text en descripciones largas).

- Motivación: Mejorar la eficiencia y escalabilidad de la base de datos.

Versión 1.3: Integración con Escaneo QR

- Fecha: 22 de octubre de 2024
- Cambios:
    - Creación de la tabla validacion_qr para registrar los códigos QR generados y escaneados.
    - Relación entre validacion_qr, asistencia y ticket.
- Motivación: Soportar la funcionalidad de validación de asistencia mediante códigos QR.

Versión 1.4: Implementación de Categorías de Tickets

- Fecha: 24 de octubre de 2024
- Cambios:
    - Creación de la tabla categoria_ticket.
    - Asociación de categoria_ticket con ticket para permitir múltiples tipos de tickets por evento.
- Motivación: Ofrecer una mayor flexibilidad en la gestión de tickets y sus categorías.

Versión 1.5: Migración a PostgreSQL en la Nube

- Fecha: 29 de octubre de 2024

- Cambios:
    - Migración de la base de datos local a un entorno PostgreSQL en la nube.
    - Ajustes en scripts de migración para adaptarse al nuevo entorno.
- Motivación: Garantizar disponibilidad y escalabilidad del sistema

II. IV. IV. Prácticas para une Versionado Efectivo

Para mantener la integridad del modelo de datos y facilitar su evolución, se adoptaron las siguientes prácticas:

1. Estándares de Migración:
    - Cada cambio debe ir acompañado de un script de migración (migraciones y rollback).
    - Los scripts se versionan en el repositorio Git del proyecto.
2. Sincronización con el Código:
    - El esquema de la base de datos está sincronizado con el backend mediante herramientas de ORM (como Sequelize o Knex).
3. Pruebas Automatizadas:
    - Cada migración incluye pruebas unitarias para validar la estructura y datos de prueba.
4. Documentación Continua:
    - Se actualiza el diagrama E-R y el diccionario de datos con cada versión.

II. V. Documentación de la arquitectura y diseño del software

El siguiente punto es un componente esencial que tiene como objetivo proporcionar una visión clara y detallada del modelo arquitectónico, los diversos componentes del sistema y su integración. Este documento es fundamental para garantizar que todos los equipos técnicos y de negocio comprendan cómo está diseñado el sistema, facilitando tanto el desarrollo continuo como el mantenimiento. A continuación, se describe detalladamente cada uno de los aspectos que conforman esta sección.

II. V. I. Definición de la Arquitectura

La arquitectura de RegCon fue seleccionada basándose en los siguientes principios clave: modularidad, escalabilidad, flexibilidad, y facilidad de mantenimiento. La decisión de adoptar una arquitectura cliente-servidor moderna con un enfoque basado en API RESTful asegura que el sistema pueda satisfacer tanto los requerimientos actuales como los futuros.

Razones para la selección de la arquitectura:

1. Separación de Responsabilidades:
    - La arquitectura cliente-servidor separa claramente la lógica de presentación (frontend) de la lógica de negocio y de datos (backend). Esto facilita el trabajo en equipo, ya que permite que los desarrolladores frontend y backend trabajen de forma independiente.
2. Escalabilidad:
    - El servidor, construido con Node.js y Express, es altamente escalable gracias a su arquitectura basada en eventos y su capacidad de manejar múltiples solicitudes concurrentes.

- La base de datos PostgreSQL, alojada en la nube, soporta tanto escalabilidad horizontal como vertical, permitiendo manejar grandes volúmenes de datos.

3. Flexibilidad para Integraciones:
    - El uso de una API RESTful facilita la integración con servicios de terceros, como proveedores de pago (Stripe, PayPal) y herramientas de análisis (Google Analytics).
4. Compatibilidad Multiplataforma:
    - El diseño basado en estándares web asegura que el sistema pueda ser accedido desde cualquier dispositivo con un navegador moderno.
5. Seguridad:
    - La arquitectura incluye medidas de seguridad robustas, como autenticación basada en JWT, cifrado de contraseñas con bcrypt, y protección de datos en tránsito mediante HTTPS.

Peculiaridades de la arquitectura:

- La introducción de grupos de trabajo como entidades independientes asegura la segmentación de datos para cada administrador, lo que permite un control granular y personalizado.
- Se eligió PostgreSQL debido a su capacidad para manejar relaciones complejas, lo cual es esencial para las funcionalidades de RegCon, como la asociación de usuarios, eventos, tickets y asistencia.

II. V. II. Diseño de Software

El diseño de software detalla el modelo arquitectónico y los componentes que forman parte del sistema. RegCon utiliza un enfoque modular para garantizar la claridad y simplicidad en el desarrollo.

Componentes Principales del Diseño:

1. Frontend:
    - Lenguaje: React para el desarrollo del cliente.
    - Componentes: Diseñados con modularidad y reutilización en mente.
        - Ejemplo: Componentes para formularios de registro, paneles de control, y escaneo de QR.
    - Gestión de Estado: Se utilizó Context API para manejar estados globales, como la sesión del usuario y los datos del grupo de trabajo.
    - Rutas: react-router-dom para la navegación y la protección de rutas según el nivel de autenticación.
2. Backend:
    - Lenguaje: Node.js con Express.
    - Estructura:
        - Controladores para manejar solicitudes HTTP.
        - Servicios que encapsulan la lógica de negocio.
        - Middlewares para tareas comunes, como autenticación y validación de datos.

- API RESTful: Proporciona una interfaz clara para la comunicación entre el cliente y el servidor.

3. Base de Datos:
    - Modelo Relacional: PostgreSQL para gestionar las relaciones entre entidades como usuarios, eventos, tickets y asistencia.
    - Optimización: Uso de índices y consultas optimizadas para mejorar el rendimiento.
4. Integración de Servicios:
    - Integración con herramientas externas para pagos, validación y análisis de datos.

Diagrama de Flujo General del Sistema: Un diagrama describe el flujo de datos desde el cliente hacia el servidor, pasando por la API RESTful, hasta la base de datos, y de vuelta hacia el cliente.

II. V. III. Definición de Interfaz de Usuario

La interfaz de usuario (UI) de RegCon fue diseñada tomando como base los principios de experiencia de usuario (UX), priorizando la usabilidad, accesibilidad y navegación intuitiva.

Elementos Principales de la UI:

1. Diseño Responsivo:
    - Compatible con dispositivos móviles, tabletas y escritorios.
    - Uso de CSS y frameworks como Tailwind o Bootstrap para asegurar la responsividad.
2. Navegación Intuitiva:

- Barra de navegación con acceso rápido a secciones importantes como Dashboard, Eventos, y Tickets.
- Botones y enlaces organizados por prioridad de uso.

3. Comportamientos Definidos:
    - Feedback visual para acciones del usuario (por ejemplo, cambios de color en botones al hacer clic o formularios con validación dinámica).
    - Confirmaciones para acciones críticas, como eliminar registros o finalizar compras.
4. Estandarización Visual:
    - Uso de colores, tipografías y elementos gráficos consistentes en toda la aplicación.
    - Íconos descriptivos para mejorar la comprensión de las funcionalidades.

Flujo de Usuario Típico:

1. Un administrador inicia sesión y accede a su panel de control.
2. Desde el panel, gestiona eventos, crea tickets y monitorea la asistencia.
3. Los asistentes, por su parte, pueden registrarse y validar su entrada mediante escaneo de QR.

II. V. IV. Mantenimiento de la Documentación

La documentación de la arquitectura y diseño del software es un recurso vivo que se actualiza con cada iteración del sistema. Esto asegura que refleje siempre el estado actual del proyecto.

Plan Final

Prácticas Implementadas:

1. Versionado de Documentación:
    - La documentación se almacena en un repositorio Git, junto con el código fuente del sistema.
    - Cada cambio en la arquitectura o el diseño se registra como un "commit" con descripciones detalladas.
2. Actualización Periódica:
    - Cada nueva funcionalidad incluye la actualización de los diagramas y descripciones asociadas.
3. Uso de Herramientas:
    - Supabase: Para mantener el diagrama entidad-relación actualizado.
    - Lucidchart: Para diagramas de flujo y diseño general.
    - Markdown o LaTeX: Para generar documentación técnica legible.

II. V. V. Beneficios de la documentación

- Facilidad de Comprensión:
    - La documentación detallada permite a los nuevos desarrolladores comprender rápidamente cómo funciona el sistema.
- Estandarización:
    - Garantiza que todos los miembros del equipo sigan un enfoque consistente para realizar cambios o agregar nuevas funcionalidades.
- Mantenibilidad:
    - Reduce el tiempo necesario para depurar o actualizar el sistema.

III. Construcción del Software

La construcción del software es el núcleo del proceso de desarrollo de RegCon, donde se implementan las funcionalidades planificadas, se estructuran los componentes del sistema, y se integran las tecnologías seleccionadas para dar vida al proyecto. Este punto aborda las prácticas, herramientas y metodologías utilizadas durante el desarrollo para garantizar un producto robusto, mantenible, y alineado con los requerimientos establecidos. El proceso de construcción de RegCon fue diseñado para asegurar calidad y escalabilidad, aplicando principios de ingeniería de software moderna. Incluye la selección de tecnologías adecuadas, la implementación de control de versiones, el desarrollo de código reutilizable, y la realización de pruebas exhaustivas. Además, se incorporaron prácticas de programación segura y un enfoque estructurado para el manejo de errores, con el objetivo de minimizar vulnerabilidades y asegurar una experiencia de usuario confiable. Cada etapa del desarrollo fue documentada cuidadosamente, desde la elección de herramientas hasta la implementación de componentes y la validación del sistema mediante pruebas unitarias e integradas. Esto no solo garantiza la sostenibilidad del sistema, sino que también facilita la incorporación de nuevos desarrolladores y la evolución del software en futuras iteraciones. En los apartados siguientes, se describen en detalle los aspectos más relevantes de la construcción del software, desglosados en nueve subpuntos clave que abarcan desde la selección de tecnologías hasta la documentación de los componentes desarrollados. Este enfoque proporciona una visión completa y estructurada del proceso de construcción, destacando las decisiones tomadas, las herramientas utilizadas y las mejores prácticas implementadas.

III. I. Selección de Tecnologías

La selección de tecnologías para el desarrollo de RegCon fue un proceso crítico, ya que debía garantizar que el sistema cumpliera con los requisitos funcionales y no funcionales, como escalabilidad, seguridad, facilidad de mantenimiento y compatibilidad multiplataforma. Para

lograr esto, se evaluaron múltiples herramientas, lenguajes y frameworks, seleccionando aquellos que mejor se alineaban con los objetivos del proyecto y las necesidades del equipo de desarrollo.

A continuación, se presenta una descripción detallada de las tecnologías seleccionadas, organizadas por capas y componentes del sistema.

III. I. I. Frontend

El frontend de RegCon fue diseñado para proporcionar una experiencia de usuario interactiva, responsiva y moderna. Para ello, se seleccionaron las siguientes tecnologías:

- React:
    - Motivación: React es una biblioteca de JavaScript ampliamente utilizada para construir interfaces de usuario dinámicas. Su enfoque basado en componentes facilita la reutilización del código y la modularidad.
    - Ventajas:
        - Capacidad para manejar estados complejos de la aplicación mediante Context API.
        - Su sistema de enrutamiento, implementado con react-router-dom, permite gestionar rutas protegidas y dinámicas.
        - Amplia comunidad y ecosistema de bibliotecas adicionales, como Material-UI o TailwindCSS.
- CSS y Frameworks de Estilo:
    - TailwindCSS: Se seleccionó por su enfoque basado en clases utilitarias, lo que permite un diseño rápido y consistente en toda la aplicación.

- Ventajas: Facilita la creación de interfaces responsivas, minimizando la duplicación de estilos y mejorando la productividad.

- Axios:
  - Motivación: Para manejar la comunicación entre el cliente y el servidor, Axios fue elegido como cliente HTTP debido a su simplicidad y soporte para interceptores.
  - Ventajas:
    - Manejo eficaz de solicitudes asincrónicas.
    - Capacidad para interceptar y gestionar respuestas de errores.

III. I. II. Backend

El backend de RegCon es responsable de procesar solicitudes, manejar la lógica empresarial y comunicarse con la base de datos. Para esta capa, se seleccionaron las siguientes tecnologías:

- Node.js:
  - Motivación: Node.js es un entorno de ejecución de JavaScript basado en el motor V8 de Google. Fue elegido por su capacidad para manejar operaciones I/O de manera eficiente, especialmente en aplicaciones que requieren manejar múltiples solicitudes concurrentes.
  - Ventajas:
    - Ideal para aplicaciones en tiempo real y basadas en eventos.
    - Gran ecosistema de paquetes y módulos, gestionados a través de npm.

- Express:
  - Motivación: Express es un framework minimalista para Node.js que simplifica la creación de servidores web y APIs RESTful.

- Ventajas:
  - Flexibilidad para construir middleware personalizado.
  - Compatibilidad con herramientas de autenticación y manejo de errores.

- JWT (JSON Web Tokens):
  - Motivación: Se utilizó para la autenticación de usuarios y protección de rutas sensibles.
  - Ventajas: Proporciona un método seguro y estándar para gestionar sesiones y permisos.

- Bcrypt:
  - Motivación: Biblioteca para cifrar contraseñas, proporcionando seguridad adicional a las credenciales almacenadas.
  - Ventajas: Resistencia a ataques de fuerza bruta.

III. I. III. Base de Datos

La base de datos juega un rol crucial en el sistema, almacenando y gestionando la información relacionada con usuarios, eventos, tickets y asistencia.

- PostgreSQL:
  - Motivación: Se seleccionó PostgreSQL debido a su capacidad para manejar relaciones complejas y grandes volúmenes de datos. Es una base de datos relacional avanzada que soporta características críticas como transacciones, integridad referencial y extensibilidad.
  - Ventajas:
    - Escalabilidad horizontal y vertical.

        - Soporte nativo para índices y consultas complejas.
        - Capacidad para manejar relaciones 1:N y N:M, esenciales para el modelo de datos de RegCon.

- Herramientas de Gestión:
    - pgAdmin: Utilizado para la administración de la base de datos, como la gestión de esquemas y la ejecución de consultas SQL.
    - DBeaver: Herramienta complementaria para la visualización y documentación del modelo de datos.

III. I. IV. Entorno de Desarrollo

El entorno de desarrollo de RegCon fue diseñado para facilitar la colaboración del equipo y garantizar la calidad del código.

- Visual Studio Code (VSCode):
    - Motivación: Editor de código ampliamente utilizado por su flexibilidad, soporte para múltiples lenguajes y extensiones específicas.
    - Ventajas:
        - Extensiones para ESLint, Prettier y Git.
        - Integración con terminal para ejecutar scripts y comandos.
- Git:
    - Motivación: Sistema de control de versiones distribuido utilizado para gestionar el código fuente del proyecto.
    - Ventajas:

        - Seguimiento de cambios en el código.
        - Facilitación del trabajo en equipo mediante ramas (branches) y solicitudes de extracción (pull requests).
    - Plataforma: GitHub para alojar el repositorio del proyecto.

III. I. V. Herramientas de Pruebas

Para asegurar la calidad del software, se seleccionaron herramientas específicas para pruebas automatizadas.

- Jest:
    - Motivación: Framework de pruebas para JavaScript que permite realizar pruebas unitarias e integradas de manera sencilla.
    - Ventajas:
        - Configuración mínima.
        - Generación de reportes de cobertura de pruebas.
- Supertest:
    - Motivación: Herramienta para realizar pruebas de integración en APIs RESTful.
    - Ventajas:
        - Simulación de solicitudes HTTP para validar el comportamiento del backend.

III. I. VI. Despliegue y Producción

El despliegue de RegCon se planificó para garantizar accesibilidad y estabilidad en entornos de producción.

- Back4App (Alojamiento en la Nube):
    - Motivación: Plataforma de hosting que soporta contenedores Docker y simplifica la gestión de infraestructura.
    - Ventajas:
        - Escalabilidad automática.
        - Costos bajos para entornos pequeños.
- Docker:
    - Motivación: Permite empaquetar la aplicación con todas sus dependencias en contenedores, asegurando consistencia entre entornos.
    - Ventajas:
        - Fácil replicación del entorno de desarrollo en producción.
        - Aislamiento de servicios.
- Vercel:
    - Motivación: Permite alojar webs construidas con React utilizando GitHub como base, por lo que la forma de desplegar las aplicaciones es más fácil y rápida.
    - Ventajas:
        - Fácil despliegue de apps desde GitHub.
        - Conocimientos y experiencias previas alojando servicios webs en Vercel.

III. II. Controles de Versiones del Código

El control de versiones del código es un pilar fundamental en el desarrollo de RegCon, ya que permite gestionar de forma eficiente los cambios en el código fuente, garantizar la colaboración

entre los miembros del equipo, y mantener un historial claro de las modificaciones realizadas durante todo el ciclo de vida del proyecto. Este proceso no solo asegura que el proyecto sea escalable y sostenible, sino que también facilita la depuración, el mantenimiento y la implementación de nuevas funcionalidades de manera controlada. A continuación, se detalla el enfoque adoptado para el control de versiones del código, incluyendo las herramientas, estrategias y prácticas empleadas para garantizar la integridad y consistencia del sistema.

III. II. I. Herramientas Utilizadas

a. Git (Sistema de Control de Versiones):

- Motivación: Git es un sistema de control de versiones distribuido que permite a los desarrolladores trabajar simultáneamente en diferentes partes del proyecto sin conflictos.
- Ventajas:
    - Almacenamiento local y remoto del historial de versiones.
    - Compatibilidad con plataformas de colaboración como GitHub.
    - Capacidades avanzadas para ramas (branches) y fusiones (merges).

b. GitHub (Repositorio Remoto):

- Motivación: GitHub proporciona una plataforma para alojar el repositorio del proyecto en la nube, lo que permite la colaboración en equipo y el acceso remoto.
- Ventajas:
    - Gestión de ramas, solicitudes de extracción (pull requests) y revisiones de código.

- Integración con herramientas de CI/CD (Integración Continua/Despliegue Continuo).
- Generación de reportes y seguimiento de problemas a través del sistema de issues.

c. Extensiones de Git en VSCode:

- Motivación: Las extensiones integradas en Visual Studio Code simplifican el uso de Git mediante interfaces gráficas y atajos de teclado.
- Ventajas:
  - Visualización de cambios en tiempo real.
  - Creación rápida de ramas y confirmaciones (commits).
  - Resolución de conflictos directamente desde el editor.

III. II. II. Flujo de Trabajo

El flujo de trabajo del control de versiones en **R**egCon se diseñó siguiendo el modelo Git Flow, adaptado para garantizar un desarrollo ordenado y colaborativo. Este flujo define cómo se manejan las ramas y cómo se integran los cambios en el proyecto.

a. Ramas Principales:

1. main (Producción):
   - Contiene la versión estable y lista para producción del sistema.
   - Solo se fusionan cambios completamente validados y probados.
2. develop (Desarrollo):
   - Rama base para la implementación de nuevas funcionalidades y correcciones.

- Los desarrolladores integran sus ramas de trabajo en esta rama una vez que sus cambios han sido revisados y aprobados.

b. Ramas Secundarias:

1. feature/<nombre> (Funcionalidades):
    - Cada nueva funcionalidad se desarrolla en una rama específica que parte de develop.
    - Ejemplo: feature/login, feature/qr-validation.
2. bugfix/<nombre> (Correcciones de Errores):
    - Utilizadas para solucionar errores encontrados en develop o main.
    - Ejemplo: bugfix/form-validation, bugfix/api-endpoint.
3. hotfix/<nombre> (Correcciones Urgentes):
    - Se crean para solucionar errores críticos en main.
    - Ejemplo: hotfix/session-timeout.

c. Estrategias de Fusión (Merge):

- Solicitud de Extracción (Pull Request):
    - Cada cambio en una rama secundaria requiere una solicitud de extracción para ser fusionado en develop o main.
    - Los cambios son revisados por al menos un miembro del equipo para asegurar la calidad del código.
- Resolución de Conflictos:

- En caso de conflictos al fusionar, se realiza una revisión colaborativa para solucionarlos.

III. II. III. Reglas y Políticas de Control de Versiones

Para garantizar la consistencia y evitar errores, se definieron las siguientes políticas para el uso del sistema de control de versiones:

1. Mensajes de Commit:
    - Los mensajes deben ser claros y descriptivos, siguiendo un formato estándar:
        - [tipo]: descripción corta
        - Ejemplo: feature: añadir validación de códigos QR, fix: corregir error en el registro de eventos.
    - Tipos de commits más comunes:
        - feature (nuevas funcionalidades).
        - fix (corrección de errores).
        - docs (actualización de documentación).
        - refactor (mejoras en el código sin cambios funcionales).
2. Ramas Nombradas Consistentemente:
    - Se definió un esquema claro para nombrar ramas:
        - feature/<nombre_descriptivo>
        - bugfix/<nombre_descriptivo>
        - hotfix/<nombre_descriptivo>
3. Revisión de Código:

- Todo cambio debe ser revisado por al menos otro miembro del equipo antes de ser fusionado.
- Se utilizan herramientas de revisión de código en GitHub para comentar y sugerir mejoras.

4. Frecuencia de Commits:
    - Se recomienda realizar commits pequeños y frecuentes para facilitar el seguimiento de cambios.

III. II. IV. Herramientas Adicionales Integradas

1. CI/CD (Integración Continua y Despliegue Continuo):
    - Se integró GitHub Actions para automatizar las pruebas y despliegues del sistema.
    - Cada solicitud de extracción activa un pipeline que:
        - Ejecuta pruebas unitarias e integradas.
        - Verifica el estado de la base de datos (migraciones).
        - Genera reportes de cobertura de pruebas.
2. Issues y Proyectos de GitHub:
    - Se utiliza el sistema de issues para rastrear errores, sugerencias y tareas pendientes.
    - Cada issue está vinculado a una rama de desarrollo, asegurando un seguimiento completo desde la planificación hasta la implementación.
3. Documentación de Cambios (Changelog):

- Se mantiene un archivo CHANGELOG.md en el repositorio para documentar todas las actualizaciones y cambios realizados en el sistema.

III. II. V. Beneficios del Control de Versiones

El uso de Git y GitHub como herramientas principales de control de versiones ha proporcionado múltiples beneficios al desarrollo de RegCon:

- Colaboración Eficiente: Permite que varios desarrolladores trabajen simultáneamente sin conflictos.
- Historial de Cambios: Registra cada modificación, lo que facilita el seguimiento y depuración del código.
- Seguridad: Los cambios en main están protegidos mediante revisiones y aprobaciones obligatorias.
- Automatización: La integración con GitHub Actions asegura que el código fusionado cumple con los estándares de calidad.

III. III. Calidad del Código

Este punto es un aspecto fundamental ya que garantiza que el sistema sea legible, mantenible, escalable y eficiente. La implementación de estándares de calidad en el código no solo facilita el trabajo del equipo de desarrollo, sino que también reduce los costos de mantenimiento, disminuye la probabilidad de errores y asegura que el software cumpla con los requisitos establecidos. En este subpunto, se detallan las prácticas, herramientas, y políticas implementadas para mantener y evaluar la calidad del código durante el desarrollo de RegCon.

III. III. I. Principios Fundamentales para la Calidad del Código

El equipo de desarrollo de RegCon adoptó varios principios clave para garantizar un código de alta calidad:

Plan Final

1. Legibilidad:
    - El código debe ser fácil de leer y entender para cualquier miembro del equipo, incluso si no fue quien lo escribió.
    - Se utilizó una nomenclatura clara y descriptiva para variables, funciones y clases.
2. Modularidad:
    - El código se diseñó en módulos pequeños y reutilizables, siguiendo el principio de Single Responsibility Principle (SRP).
    - Cada módulo realiza una única función, lo que facilita la depuración y el mantenimiento.
3. Consistencia:
    - Se siguieron convenciones de estilo consistentes en todo el proyecto, alineadas con las mejores prácticas del lenguaje utilizado (JavaScript/TypeScript).
4. Escalabilidad:
    - Se diseñó el código para soportar futuras expansiones, asegurando que nuevas funcionalidades puedan integrarse sin afectar el núcleo del sistema.
5. Pruebas Automatizadas:
    - Se implementaron pruebas unitarias y de integración para validar la funcionalidad del código y evitar regresiones.

III. III. II. Herramientas para la Gestión de Calidad del Código

Para garantizar la calidad del código en RegCon, se integraron diversas herramientas que ayudaron a detectar errores, mejorar la legibilidad y mantener estándares consistentes.

1. ESLint (Linting de Código):
    - Motivación: ESLint es una herramienta de análisis estático que identifica problemas en el código JavaScript.
    - Configuración: Se utilizó una configuración personalizada basada en las reglas recomendadas por ESLint y adaptada a las necesidades del proyecto.
    - Ventajas:
        - Prevención de errores comunes en JavaScript.
        - Promoción de mejores prácticas de codificación.
        - Detección de código no utilizado o mal estructurado.
2. Prettier (Formateo de Código):
    - Motivación: Prettier asegura que el código esté formateado de manera consistente, independientemente del desarrollador que lo haya escrito.
    - Ventajas:
        - Uniformidad en el estilo del código.
        - Ahorro de tiempo al eliminar discusiones sobre formato.
        - Integración con VSCode para aplicar automáticamente el formateo al guardar.
3. SonarQube (Análisis de Código Estático):

- Motivación: SonarQube se utilizó para realizar análisis avanzados del código, evaluando métricas como complejidad ciclomática, deuda técnica y duplicación de código.
- Ventajas:
  - Identificación de áreas de mejora en el diseño del código.
  - Detección de vulnerabilidades de seguridad.
  - Reportes detallados que guían al equipo en la resolución de problemas.

4. GitHub Actions (Automatización de Calidad):
   - Motivación: GitHub Actions se configuró para ejecutar análisis de calidad automáticamente en cada push o pull request.
   - Ventajas:
     - Integración continua de herramientas como ESLint y SonarQube.
     - Evaluación en tiempo real del estado del código.
     - Bloqueo de fusiones a main si se detectan errores críticos.

III. III. III. Estandarización y Mejores Prácticas

El equipo definió un conjunto de estándares y mejores prácticas para mantener la calidad del código consistente en todo el proyecto:

1. Convenciones de Nombres:
   - Variables y funciones: Notación camelCase (nombreUsuario, calcularTotal).
   - Clases y componentes: Notación PascalCase (UserForm, EventCard).

- Constantes: Notación en mayúsculas y guiones bajos (MAX_RETRIES, API_BASE_URL).

2. Estructura del Código:

- El código se organiza en carpetas por módulos funcionales:

```
src/
├── components/
├── services/
├── utils/
├── routes/
├── middleware/
├── models/
└── config/
```

- Cada archivo contiene una única responsabilidad, siguiendo el principio "un archivo, una función principal".

3. Documentación del Código:

- Cada función y clase incluye comentarios claros que describen su propósito y uso.
- Se utiliza JSDoc para generar documentación automática del código.

4. Refactorización Periódica:

- Se revisa y mejora el código regularmente para reducir la complejidad y eliminar duplicaciones.

- La complejidad ciclomática se mantiene baja para garantizar que el código sea fácil de entender y probar.

5. Revisiones de Código:
    - Todo cambio al código pasa por un proceso de revisión en GitHub antes de fusionarse.
    - Los revisores verifican la calidad, seguridad y consistencia del código.

III. III. IV. Métricas de Calidad

El equipo utilizó métricas específicas para medir y mejorar la calidad del código a lo largo del desarrollo:

1. Complejidad Ciclomática:
    - Se monitoreó la complejidad de las funciones para evitar estructuras excesivamente complicadas.
    - Objetivo: Mantener la complejidad de las funciones por debajo de 10.
2. Cobertura de Pruebas:
    - Se midió el porcentaje de líneas de código cubiertas por pruebas unitarias e integradas.
    - Objetivo: Mantener una cobertura mínima del 85%.
3. Deuda Técnica:
    - Se calculó la deuda técnica utilizando SonarQube, priorizando la resolución de problemas críticos y mayores.
4. Duplicación de Código:

- Se utilizó SonarQube para identificar y eliminar duplicaciones de código innecesarias.

III. III. V. Ejemplo Práctico: Implementación de una Función

Función: Validar un correo electrónico

```
/**
 * Valida si un correo electrónico tiene el formato correcto.
 * @param {string} email - Correo electrónico a validar.
 * @returns {boolean} - `true` si el correo es válido, `false` en caso contrario.
 */
function validarEmail(email) {
  const regex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
  return regex.test(email);
}
```

Características del ejemplo:

- Nombres descriptivos.
- Comentarios que explican el propósito de la función y sus parámetros.
- Simplicidad en la implementación.

III. III. VI. Beneficios Obtenidos

La implementación de estándares y herramientas de calidad del código ha proporcionado múltiples beneficios:

- Reducción de Errores: Las herramientas como ESLint y SonarQube detectaron problemas antes de que llegaran a producción.
- Mantenibilidad: La modularidad y claridad del código facilitaron la incorporación de nuevas funcionalidades.
- Eficiencia en el Equipo: Las revisiones de código y el formateo automático redujeron el tiempo dedicado a corregir problemas estilísticos.
- Confiabilidad: La alta calidad del código incrementó la confianza en el sistema, tanto para los desarrolladores como para los usuarios.

III. IV. Desarrollo de Componentes Reutilizables

El desarrollo de componentes reutilizables es un aspecto clave en la construcción de RegCon, ya que promueve la eficiencia, la modularidad, y la consistencia en el sistema. La reutilización de componentes no solo mejora la productividad del equipo, sino que también reduce la duplicación de código, lo que facilita el mantenimiento y la escalabilidad del software. A continuación, se describe en detalle cómo se abordó este aspecto en el proyecto, las técnicas y herramientas utilizadas, y ejemplos prácticos de su implementación.

III. IV. I. Principios del Desarrollo de Componentes Reutilizables

Para garantizar la creación de componentes reutilizables, el equipo siguió los siguientes principios fundamentales:

1. Modularidad:
    - Cada componente debe ser independiente y realizar una única función específica, alineándose con el Single Responsibility Principle (SRP).
    - Esto permite que los componentes puedan ser utilizados en diferentes partes del sistema sin depender de elementos externos.

2. Reusabilidad:
    - Los componentes se diseñaron para ser genéricos y parametrizables, evitando dependencias rígidas.
    - Se utilizó la propiedad de configuración a través de props (en React) para personalizar el comportamiento y la apariencia de los componentes.
3. Consistencia:
    - Se crearon estándares de diseño y comportamiento para garantizar que los componentes tengan un estilo y funcionalidad coherente en toda la aplicación.
    - Esto asegura que la experiencia del usuario sea uniforme.
4. Escalabilidad:
    - Los componentes se diseñaron pensando en futuras expansiones, permitiendo agregar nuevas funcionalidades sin afectar el código existente.
5. Separación de Preocupaciones:
    - Se evitó mezclar lógica de negocio con la presentación dentro de los componentes, dividiendo estas responsabilidades entre servicios y componentes.

III. IV. II. Herramientas Utilizadas para Desarrollar Componentes Reutilizables

1. React (Librería Principal):

- Motivación: React es ideal para crear interfaces de usuario basadas en componentes reutilizables.
- Características:

- Uso de hooks como useState y useEffect para manejar el estado y efectos secundarios dentro de los componentes.
- PropTypes para validar las propiedades que recibe cada componente, garantizando su correcto uso.

2. Storybook:

- Motivación: Herramienta para documentar y probar componentes de manera aislada.
- Ventajas:
  - Permite visualizar y probar los componentes sin necesidad de cargar toda la aplicación.
  - Facilita la creación de una biblioteca de componentes reutilizables accesible para todo el equipo.

3. TailwindCSS:

- Motivación: Framework de CSS que utiliza clases utilitarias, lo que facilita la creación de estilos reutilizables.
- Ventajas:
  - Evita la duplicación de estilos entre componentes.
  - Facilita la personalización de componentes mediante la combinación de clases.

4. React Context API:

- Motivación: Herramienta para compartir estados globales entre componentes sin necesidad de prop drilling.
- Ventajas:

- Permite que los componentes reutilizables accedan a datos globales como autenticación, tema o configuraciones.

III. IV. III. Categorías de Componentes Reutilizables en RegCon

Los componentes reutilizables desarrollados en RegCon se organizaron en tres categorías principales:

a. Componentes de Presentación (UI):

- Diseñados para mostrar elementos visuales sin contener lógica de negocio.
- Ejemplos:
  - Botón (Button):
    - Componente genérico que soporta diferentes estilos (primario, secundario, deshabilitado) y tamaños.
    - Props: label, onClick, type, className.
    - Ejemplo de uso:

```
<Button label="Guardar" type="submit" onClick={handleSave} />
```

  - Tabla Dinámica (DynamicTable):
    - Renderiza tablas configurables con soporte para paginación, ordenamiento y acciones personalizables.
    - Props: data, columns, onRowClick.
    - Ejemplo de uso:

```
<DynamicTable
  data={events}
```

```
  columns={['Nombre', 'Fecha', 'Estado']}
  onRowClick={handleRowClick}
/>
```

b. Componentes Funcionales:

- Encapsulan lógica que se utiliza en múltiples partes de la aplicación.
- Ejemplos:
    - Formulario de Entrada (InputField):
        - Campo de entrada genérico que soporta validaciones y mensajes de error.
        - Props: label, value, onChange, error.
        - Ejemplo de uso:

```
<InputField
  label="Correo Electrónico"
  value={email}
  onChange={handleEmailChange}
  error={emailError}
/>
```

    - Modal (Modal):
        - Componente para mostrar ventanas emergentes con contenido personalizable.

- Props: isOpen, onClose, title, children.
- Ejemplo de uso:

```
<Modal isOpen={showModal} onClose={handleClose} title="Confirmar">
  <p>¿Estás seguro de eliminar este evento?</p>
</Modal>
```

c. Componentes de Navegación:

- Encargados de gestionar la estructura y la navegación de la aplicación.
- Ejemplos:
  - Barra de Navegación (Navbar):
    - Componente reutilizable para la navegación principal, con enlaces dinámicos según el estado de autenticación.
  - Rutas Protegidas (ProtectedRoute):
    - Componente que verifica si un usuario está autenticado antes de permitir el acceso a una ruta específica.
    - Props: isAuthenticated, children.

III. IV. IV. Ejemplo Práctico de Reutilización

Componente Genérico de Botón:

```
import PropTypes from 'prop-types';

export const Button = ({ label, type, onClick, className }) => {
  return (
```

```
    <button
      type={type}
      onClick={onClick}
      className={`btn ${className}`}
    >
      {label}
    </button>
  );
};

Button.propTypes = {
  label: PropTypes.string.isRequired,
  type: PropTypes.string,
  onClick: PropTypes.func,
  className: PropTypes.string,
};

Button.defaultProps = {
  type: 'button',
  className: '',
```

};

Uso del Componente en Diferentes Contextos:

<Button label="Guardar" type="submit" className="btn-primary" />

<Button label="Cancelar" type="button" onClick={handleCancel} className="btn-secondary" />

III. IV. V. Beneficios del Desarrollo de Componentes Reutilizables

1. Eficiencia en el Desarrollo:
    - Reducir el tiempo necesario para implementar nuevas funcionalidades al reutilizar componentes existentes.
2. Mantenibilidad:
    - Cualquier cambio o mejora en un componente se refleja automáticamente en todas las partes de la aplicación que lo utilizan.
3. Consistencia Visual y Funcional:
    - Los componentes reutilizables aseguran que la experiencia del usuario sea coherente en toda la aplicación.
4. Escalabilidad:
    - Facilita la integración de nuevas funcionalidades sin necesidad de modificar componentes existentes.

III. V. Pruebas Unitarias

Las pruebas unitarias son una parte esencial en el desarrollo de RegCon, ya que aseguran que cada componente del sistema funcione correctamente de manera independiente. Este tipo de pruebas se centra en validar pequeñas unidades de código, como funciones, métodos o componentes individuales, garantizando que cumplan con su propósito sin errores.

El enfoque hacia las pruebas unitarias en RegCon permitió identificar y corregir errores de manera temprana, lo que contribuyó significativamente a la calidad del software y redujo los costos de mantenimiento. A continuación, se describen en detalle los objetivos, herramientas utilizadas, estrategias implementadas y ejemplos prácticos de las pruebas unitarias realizadas en el proyecto.

III. V. I. Objetivos de las Pruebas Unitarias

El principal objetivo de las pruebas unitarias es garantizar que cada unidad de código funcione correctamente y cumpla con los requisitos específicos. Esto se desglosa en los siguientes objetivos clave:

1. Validación del Comportamiento de Componentes:
    - Asegurar que los componentes y funciones individuales produzcan los resultados esperados bajo diferentes condiciones.
2. Detección Temprana de Errores:
    - Identificar errores en la lógica de negocio, cálculos, validaciones o flujos antes de integrar el componente con el resto del sistema.
3. Prevención de Regresiones:
    - Verificar que los cambios realizados en el código no rompan funcionalidades previamente implementadas.
4. Facilitación del Refactorizado:
    - Permitir cambios en el código con confianza, sabiendo que las pruebas unitarias validarán que el comportamiento esperado se mantiene.
5. Automatización de Pruebas:

- Reducir la dependencia de pruebas manuales mediante la implementación de pruebas automatizadas.

III. V. II. Herramientas Utilizadas

Para implementar y ejecutar las pruebas unitarias en RegCon, se seleccionaron herramientas especializadas que se integran eficientemente con el stack tecnológico del proyecto:

1. Jest:
    - Motivación: Framework de pruebas para JavaScript diseñado para realizar pruebas unitarias rápidas y sencillas.
    - Características:
        - Permite realizar pruebas asíncronas.
        - Proporciona un entorno de simulación para funciones, módulos y dependencias.
        - Genera reportes de cobertura automáticamente.
    - Ventajas:
        - Configuración mínima.
        - Integración nativa con React y Node.js.
        - Amplia comunidad y documentación.
2. Testing Library (React Testing Library):
    - Motivación: Librería enfocada en probar componentes React desde la perspectiva del usuario.
    - Características:

        - Simula la interacción del usuario con la interfaz.
        - Se centra en cómo los componentes se comportan y no en su implementación interna.
    - Ventajas:
        - Facilita pruebas basadas en accesibilidad.
        - Promueve buenas prácticas de diseño de componentes.

3. Mocks y Stubs:
    - Se utilizaron herramientas como jest.fn() para simular dependencias o funciones externas, lo que permite probar componentes aislados de su contexto.
4. GitHub Actions:
    - Motivación: Automatización de la ejecución de pruebas unitarias en cada solicitud de extracción (pull request).
    - Ventajas:
        - Asegura que el código que se fusiona en main cumpla con los requisitos establecidos.
        - Reporta fallos directamente en la plataforma.

III. V. III. Estrategia de Pruebas Unitarias

El equipo adoptó un enfoque sistemático para diseñar y ejecutar pruebas unitarias, asegurando que cada caso relevante fuera cubierto:

1. Cobertura Exhaustiva:

- Las pruebas se diseñaron para cubrir todos los posibles escenarios de uso, incluyendo:
  - Casos de uso normales.
  - Casos de borde (edge cases).
  - Comportamiento ante entradas inválidas o inesperadas.

2. Pruebas Aisladas:
   - Cada prueba se centra en una unidad de código específica, sin depender de otros módulos o servicios.
   - Se utilizó la técnica de mocking para simular dependencias externas, como llamadas a la API o bases de datos.
3. Automatización:
   - Todas las pruebas unitarias se integraron en el pipeline de CI/CD, garantizando que se ejecuten automáticamente en cada cambio de código.
4. Revisión de Resultados:
   - Los desarrolladores revisan regularmente los resultados de las pruebas para identificar patrones de errores y mejorar el diseño del código.

III. V. IV. Pruebas Unitarias en RegCon

a. Pruebas de Funciones:

- Función: Validar un correo electrónico

```
// Archivo: utils/validation.js

export const validarEmail = (email) => {
```

```
  const regex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;

  return regex.test(email);

};

// Prueba unitaria: validation.test.js

import { validarEmail } from './validation';

describe('Función validarEmail', () => {

  test('Debería retornar true para un correo válido', () => {

    expect(validarEmail('usuario@ejemplo.com')).toBe(true);

  });

  test('Debería retornar false para un correo inválido', () => {

    expect(validarEmail('usuario@ejemplo')).toBe(false);

  });

  test('Debería retornar false para una cadena vacía', () => {

    expect(validarEmail('')).toBe(false);

  });

});
```

b. Pruebas de Componentes React:

- Componente: Botón Genérico

```
// Archivo: components/Button.js

import React from 'react';
```

```
export const Button = ({ label, onClick }) => (
  <button onClick={onClick}>{label}</button>
);
// Prueba unitaria: Button.test.js
import { render, fireEvent } from '@testing-library/react';
import { Button } from './Button';
describe('Componente Button', () => {
  test('Debería renderizar el label correctamente', () => {
    const { getByText } = render(<Button label="Guardar" />);
    expect(getByText('Guardar')).toBeInTheDocument();
  });
  test('Debería llamar a la función onClick al hacer clic', () => {
    const mockOnClick = jest.fn();
    const { getByText } = render(<Button label="Guardar" onClick={mockOnClick} />);
    fireEvent.click(getByText('Guardar'));
    expect(mockOnClick).toHaveBeenCalledTimes(1);
  });
});
```

III. V. V. Cobertura de Pruebas

Se midió la cobertura de pruebas utilizando Jest, enfocándose en cuatro métricas principales:

1. Cobertura de Líneas: Verifica que cada línea de código sea ejecutada al menos una vez durante las pruebas.
2. Cobertura de Funciones: Asegura que todas las funciones sean invocadas.
3. Cobertura de Ramas: Valida que todas las ramas condicionales sean ejecutadas.
4. Cobertura de Archivos: Evalúa que cada archivo relevante esté siendo probado.

Meta del Proyecto: Mantener una cobertura mínima del 85% en todas las métricas mencionadas.

III. V. VI. Beneficios de las Pruebas Unitarias

1. Detección Temprana de Problemas:
   - Los errores se identificaron en etapas iniciales del desarrollo, reduciendo costos de corrección.
2. Prevención de Regresiones:
   - Los cambios en el código no afectaron funcionalidades existentes gracias a las pruebas automatizadas.
3. Facilitación del Refactorizado:
   - Las pruebas unitarias proporcionaron confianza para realizar refactorizaciones sin introducir nuevos errores.
4. Ahorro de Tiempo:
   - La ejecución automática de pruebas ahorró tiempo en comparación con las pruebas manuales.

III. VI. Pruebas de Integración

Las pruebas de integración son un componente esencial en el desarrollo de RegCon, diseñadas para validar cómo interactúan los diferentes módulos y componentes del sistema entre sí. A diferencia de las pruebas unitarias, que se centran en unidades individuales de código, las pruebas de integración verifican que estas unidades funcionen correctamente cuando se combinan. En RegCon, estas pruebas fueron especialmente importantes para garantizar que las interacciones entre el frontend, backend y la base de datos se ejecutaran de manera fluida y sin errores. El enfoque estructurado hacia las pruebas de integración permitió detectar problemas en la interacción entre componentes antes de que el sistema fuera desplegado en producción. Esto resultó en un software más robusto y confiable, capaz de manejar casos de uso complejos y datos en tiempo real.

III. VI. I. Objetivos de las Pruebas de Integración

El objetivo principal de las pruebas de integración es garantizar la funcionalidad correcta de los flujos de trabajo completos del sistema. Algunos objetivos clave incluyen:

1. Validación de Interacciones:
    - Garantizar que los componentes individuales (por ejemplo, frontend y backend) se comuniquen correctamente a través de la API REST.
2. Integridad de Datos:
    - Asegurar que los datos enviados desde el frontend se procesen correctamente en el backend y se almacenen adecuadamente en la base de datos.
3. Detección de Problemas de Comunicación:
    - Identificar problemas de configuración, como errores en endpoints, tiempo de espera, o incompatibilidades en formatos de datos (JSON).

4. Flujos Complejos:
    - Validar que los flujos completos del sistema, como el registro de un usuario, la compra de tickets o la validación de asistencia mediante QR, funcionen de extremo a extremo.
5. Preparación para Entornos Reales:
    - Simular entornos de producción para asegurar que las integraciones funcionen bajo condiciones reales.

III. VI. II. Herramientas Utilizadas

Para realizar pruebas de integración en RegCon, se seleccionaron herramientas especializadas que soportan pruebas de extremo a extremo y simulación de interacciones complejas:

1. Supertest:
    - Motivación: Herramienta utilizada para probar APIs RESTful en el backend de manera automatizada.
    - Características:
        - Simula solicitudes HTTP (GET, POST, PUT, DELETE) hacia los endpoints del servidor.
        - Valida respuestas como códigos de estado HTTP, encabezados y cuerpos de respuesta.
    - Ventajas:
        - Ideal para probar la interacción entre el backend y la base de datos.
2. Postman:

- Motivación: Herramienta gráfica para probar y automatizar solicitudes API de extremo a extremo.
- Características:
  - Permite la ejecución de pruebas manuales y automatizadas para los endpoints.
  - Soporte para manejar autenticación, variables globales y scripts predefinidos.
- Ventajas:
  - Útil para pruebas rápidas y para documentar los endpoints de la API.

3. Jest (Pruebas Asíncronas):
   - Motivación: Utilizado junto con Supertest para estructurar y ejecutar pruebas automatizadas de integración.
   - Características:
     - Compatible con promesas y funciones asíncronas.
     - Proporciona reportes detallados sobre el estado de las pruebas.
4. Docker (Entorno de Pruebas):
   - Motivación: Se utilizó Docker para crear entornos de prueba consistentes, asegurando que las pruebas de integración se ejecutaran en un entorno controlado.
   - Ventajas:
     - Simulación de la base de datos PostgreSQL en un contenedor.

- Reproducción exacta de los entornos de desarrollo y producción

III. VI. III. Estrategia de Pruebas de Integración

El equipo de desarrollo de RegCon adoptó una estrategia sistemática para diseñar y ejecutar pruebas de integración. Esta estrategia incluyó los siguientes pasos:

1. Identificación de Componentes Clave:
    - Se priorizaron las interacciones críticas del sistema, como:
        - Registro de usuarios.
        - Autenticación y autorización mediante JWT.
        - Compra y asignación de tickets.
        - Validación de asistencia mediante escaneo de códigos QR.
2. Simulación de Escenarios Reales:
    - Se diseñaron pruebas basadas en escenarios reales de uso, incluyendo:
        - Usuarios ingresando datos incorrectos o incompletos.
        - Errores de red o fallos de conexión.
        - Carga de datos grandes en solicitudes POST.
3. Automatización de Pruebas:
    - Todas las pruebas de integración se incluyeron en el pipeline de CI/CD para ejecutarse automáticamente en cada cambio del código.
4. Validación de Datos:
    - Se verificó que los datos procesados por el backend coincidieran con los esperados en la base de datos y el frontend.

Plan Final

III. VI. IV. Pruebas de Integración

a. Prueba de Registro de Usuario

- Descripción: Validar que el flujo de registro de un nuevo usuario funcione correctamente.
- Prueba:

```
import request from 'supertest';

import app from '../app'; // Importa el servidor de la aplicación

describe('Pruebas de integración: Registro de usuario', () => {

  test('Debería registrar un nuevo usuario y retornar un token', async () => {

    const response = await request(app)

      .post('/api/users/register')

      .send({

        nombre: 'Usuario Prueba',

        email: 'usuario@prueba.com',

        password: '12345678',

      });

    expect(response.status).toBe(201);

    expect(response.body).toHaveProperty('token');

    expect(response.body.token).toBeDefined();

  });
```

```
  test('Debería fallar si el email ya está registrado', async () => {
    const response = await request(app)
      .post('/api/users/register')
      .send({
        nombre: 'Usuario Prueba',
        email: 'usuario@prueba.com',
        password: '12345678',
      });
    expect(response.status).toBe(400);
    expect(response.body).toHaveProperty('error');
  });
});
```

b. Prueba de Validación de Asistencia (Códigos QR)

- Descripción: Simular la validación de un ticket mediante el escaneo de un código QR.
- Prueba:

```
describe('Pruebas de integración: Validación de códigos QR', () => {
  test('Debería validar correctamente un código QR válido', async () => {
    const response = await request(app)
      .post('/api/attendance/validate')
      .send({ qrCode: 'validQRCode123' });
```

```
    expect(response.status).toBe(200);
    expect(response.body).toHaveProperty('message', 'Asistencia validada correctamente');
  });
  test('Debería fallar con un código QR inválido', async () => {
    const response = await request(app)
      .post('/api/attendance/validate')
      .send({ qrCode: 'invalidQRCode' });
    expect(response.status).toBe(400);
    expect(response.body).toHaveProperty('error', 'Código QR no válido');
  });
});
```

III. VI. V. Cobertura de Pruebas

Para medir la efectividad de las pruebas de integración, se analizaron las siguientes métricas:

1. Cobertura de Endpoints:
    - Se aseguró que todos los endpoints críticos de la API estuvieran cubiertos por pruebas.
2. Cobertura de Flujos Complejos:
    - Validación de procesos que involucran múltiples módulos, como la compra de tickets y su validación.
3. Tiempos de Respuesta:

- Se monitorearon los tiempos de respuesta de las solicitudes para garantizar el rendimiento esperado.

III. VI. VI. Beneficios de las Pruebas de Integración

1. Confiabilidad del Sistema:
    - Aseguraron que los módulos interactúen correctamente bajo diferentes escenarios.
2. Reducción de Riesgos:
    - Identificación temprana de problemas de integración que podrían haber afectado a los usuarios finales.
3. Mejoras en el Rendimiento:
    - Verificación de tiempos de respuesta óptimos en interacciones clave.
4. Facilidad para Escalar:
    - Al validar la comunicación entre módulos, se garantiza que el sistema pueda soportar futuras expansiones.

III. VII. Prácticas de Programación Segura

La seguridad es un componente crítico en el desarrollo de RegCon, especialmente considerando que el sistema maneja datos sensibles como información personal de usuarios, detalles de eventos, tickets y asistencia. Para garantizar la protección de los datos y prevenir vulnerabilidades, se implementaron una serie de prácticas de programación segura en todas las etapas del desarrollo. Estas prácticas abarcan desde el diseño del sistema hasta la implementación y el despliegue, asegurando que el software cumpla con los estándares de seguridad modernos y sea resistente a amenazas comunes.

A continuación, se detallan las prácticas de programación segura adoptadas en el proyecto RegCon, junto con las herramientas y técnicas específicas implementadas para cada una.

III. VII. I. Principios Fundamentales de Programación Segura

1. Defensa en Profundidad:
    - Se implementaron múltiples capas de seguridad para proteger los datos en tránsito, en reposo y durante el procesamiento.
2. Menor Privilegio:
    - Cada componente, usuario y proceso tiene acceso únicamente a los recursos que necesita para funcionar, minimizando el impacto de posibles brechas.
3. Validación y Sanitización de Datos:
    - Todos los datos ingresados por el usuario son validados y sanitizados antes de ser procesados, previniendo inyecciones de código y otras amenazas.
4. Cifrado de Datos Sensibles:
    - Las contraseñas y datos sensibles son cifrados utilizando algoritmos robustos, como bcrypt y TLS para el tráfico de red.
5. Actualizaciones y Parches:
    - Las dependencias y bibliotecas utilizadas se mantienen actualizadas para proteger el sistema contra vulnerabilidades conocidas.

III. VII. II. Prácticas de Seguridad Implementadas en RegCon

a. Validación de Entradas

- Descripción:

- Todas las entradas proporcionadas por los usuarios, como formularios de registro, inicio de sesión o creación de eventos, son validadas para garantizar que cumplen con los formatos esperados.

- Implementación:
  - Se utilizaron bibliotecas como Joi para definir esquemas de validación en el backend.
  - Ejemplo:

```
const Joi = require('joi');


const userSchema = Joi.object({
  nombre: Joi.string().min(3).max(100).required(),
  email: Joi.string().email().required(),
  password: Joi.string().min(8).required(),
});


const validarUsuario = (req, res, next) => {
  const { error } = userSchema.validate(req.body);
  if (error) {
    return res.status(400).json({ error: error.details[0].message });
  }
```

```
  next();
};
```

- Beneficio: Previene ataques como inyección de SQL o envío de datos malformados.

b. Gestión de Sesiones y Autenticación

- Descripción:
    - La autenticación de usuarios se realiza mediante JSON Web Tokens (JWT), y las sesiones se manejan de forma segura.
- Implementación:
    - Los tokens se generan con una clave secreta robusta y tienen una expiración limitada.
    - Ejemplo:

```
const jwt = require('jsonwebtoken');


const generarToken = (userId) => {
  return jwt.sign({ id: userId }, process.env.JWT_SECRET, { expiresIn: '1h' });
};
```

    - Las rutas protegidas verifican la validez del token antes de permitir el acceso:

```
const verificarToken = (req, res, next) => {
  const token = req.headers['authorization'];
  if (!token) return res.status(401).json({ error: 'Acceso denegado' });
```

```
  jwt.verify(token, process.env.JWT_SECRET, (err, decoded) => {
    if (err) return res.status(403).json({ error: 'Token inválido' });
    req.userId = decoded.id;
    next();
  });
};
```

- Beneficio: Garantiza que solo los usuarios autenticados puedan acceder a recursos sensibles.
-

c. Cifrado de Contraseñas

- Descripción:
    - Las contraseñas de los usuarios se cifran antes de almacenarse en la base de datos, utilizando algoritmos robustos.
- Implementación:
    - Se utilizó la biblioteca bcrypt para cifrar las contraseñas.
    - Ejemplo:

```
const bcrypt = require('bcrypt');


const cifrarPassword = async (password) => {
```

```
  const salt = await bcrypt.genSalt(10);

  return await bcrypt.hash(password, salt);

};



const compararPassword = async (password, hashedPassword) => {

  return await bcrypt.compare(password, hashedPassword);

};
```

- Beneficio: Previene que las contraseñas sean expuestas incluso si la base de datos es comprometida.

d. Protección contra Inyección SQL

- Descripción:
    - Las consultas a la base de datos utilizan parámetros preparados para evitar inyecciones de SQL.
- Implementación:
    - Se utilizó Sequelize (ORM) para manejar las interacciones con la base de datos.
    - Ejemplo:

```
const usuario = await Usuario.findOne({

  where: {

    email: req.body.email,

  },
```

```
});
```

- Beneficio: Previene que usuarios malintencionados ejecuten comandos SQL maliciosos.

e. Seguridad en el Tráfico de Red

- Descripción:
    - Todo el tráfico entre el cliente y el servidor está cifrado utilizando HTTPS.
- Implementación:
    - Se configuró un certificado SSL en el servidor.
    - Ejemplo:

```
const https = require('https');
const fs = require('fs');


const server = https.createServer({
  key: fs.readFileSync('key.pem'),
  cert: fs.readFileSync('cert.pem'),
}, app);


server.listen(443, () => {
  console.log('Servidor HTTPS corriendo en el puerto 443');
});
```

- Beneficio: Protege los datos en tránsito contra ataques de interceptación como man-in-the-middle.

f. Protección Contra Ataques de Fuerza Bruta

- Descripción:
    - Se limitaron las solicitudes de autenticación por usuario para prevenir ataques de fuerza bruta.
- Implementación:
    - Se utilizó la biblioteca express-rate-limit.
    - Ejemplo:

```
const rateLimit = require('express-rate-limit');


const limiteLogin = rateLimit({
  windowMs: 15 * 60 * 1000, // 15 minutos
  max: 5, // máximo de 5 intentos por IP
  message: 'Demasiados intentos de inicio de sesión. Por favor, inténtelo más tarde.',
});


app.use('/api/users/login', limiteLogin);
```

- Beneficio: Evita intentos masivos de acceso no autorizado.

g. Logs y Monitoreo de Seguridad

- Descripción:
    - Se implementaron herramientas de registro y monitoreo para detectar actividades sospechosas en el sistema.
- Implementación:
    - Uso de bibliotecas como Winston para registrar eventos importantes:

```
const winston = require('winston');


const logger = winston.createLogger({
  level: 'info',
  format: winston.format.json(),
  transports: [
    new winston.transports.File({ filename: 'logs/error.log', level: 'error' }),
    new winston.transports.Console(),
  ],
});
```

- Beneficio: Facilita la detección y análisis de ataques o errores.

III. VII. III. Beneficios de las Prácticas de Programación Segura

1. Prevención de Amenazas Comunes:
    - Protección contra inyección de SQL, ataques XSS y CSRF.
    - Reducción del riesgo de exposición de datos sensibles.

2. Cumplimiento de Normas:

    - Garantía de que el sistema cumple con regulaciones de protección de datos como GDPR y CCPA.

3. Confianza del Usuario:

    - Los usuarios confían en el sistema sabiendo que sus datos están protegidos.

4. Reducción de Riesgos:

    - Minimización del impacto de posibles vulnerabilidades o brechas de seguridad.

III. VIII. Manejo de Errores

El manejo de errores es una de las características más importantes en el desarrollo de RegCon, ya que asegura que el sistema pueda responder adecuadamente a situaciones imprevistas, mantener la estabilidad y proporcionar información útil tanto al usuario como a los desarrolladores. Un sistema bien diseñado no solo maneja errores de forma eficiente, sino que también evita que estos comprometan la seguridad o la experiencia del usuario. En RegCon, el manejo de errores se implementó de manera integral en todas las capas del sistema, desde la validación de entradas del usuario en el frontend hasta el manejo de excepciones en el backend. Este enfoque asegura que los errores sean capturados, registrados y gestionados correctamente, permitiendo una resolución rápida y efectiva.

III. VIII. I. Objetivos del Manejo de Errores

1. Estabilidad del Sistema:
    - Garantizar que los errores no causen fallos en cascada ni bloqueen la operación del sistema.
2. Experiencia del Usuario:

- Proporcionar mensajes claros y útiles al usuario final cuando ocurra un error, evitando exponer detalles técnicos.

3. Monitoreo y Resolución:
    - Registrar los errores en un sistema centralizado para facilitar su análisis y resolución.
4. Seguridad:
    - Evitar que los errores expongan información sensible o permitan vulnerabilidades de seguridad.
5. Prevención de Errores Recurrentes:
    - Identificar patrones de errores para mejorar el sistema y prevenir problemas similares en el futuro.

III. VIII. II. Prácticas Implementadas para el Manejo de Errores

a. Validación de Entradas

- Descripción:
    - Antes de procesar cualquier dato ingresado por el usuario, se valida para asegurarse de que cumple con los formatos y restricciones esperados.
- Implementación:
    - Se utilizó la biblioteca Joi para validar los datos de entrada en el backend.

```
const Joi = require('joi');


const schema = Joi.object({
```

```
  email: Joi.string().email().required(),
  password: Joi.string().min(8).required(),
});


app.post('/api/login', (req, res, next) => {
  const { error } = schema.validate(req.body);
  if (error) {
    return res.status(400).json({ error: error.details[0].message });
  }
  next();
});
```

- Beneficio: Los errores relacionados con entradas inválidas se detectan temprano y se comunican al usuario de manera clara.

b. Middleware de Manejo de Errores en el Backend

- Descripción:
    - En el backend, se utiliza un middleware global para capturar y manejar errores que ocurren durante la ejecución de las solicitudes.
- Implementación:

```
app.use((err, req, res, next) => {
  console.error(err.stack); // Registro detallado del error para los desarrolladores
```

```
  if (err.name === 'ValidationError') {
    return res.status(400).json({ error: 'Datos inválidos proporcionados' });
  }

  // Respuesta genérica para errores no manejados
  res.status(500).json({ error: 'Ocurrió un error inesperado. Intente nuevamente más tarde.' });
});
```

- Beneficio: Centraliza el manejo de errores, garantizando respuestas consistentes y evitando fugas de información técnica al cliente.

c. Captura de Errores en el Frontend

- Descripción:
    - Los errores que ocurren en el frontend, como fallos en solicitudes HTTP o errores en la interfaz, son capturados y gestionados para evitar que afecten la experiencia del usuario.
- Implementación:
    - Uso de bloques try-catch y manejo de errores en promesas.

```
const login = async (email, password) => {
  try {
    const response = await axios.post('/api/login', { email, password });
```

```
    console.log('Usuario autenticado:', response.data);
  } catch (error) {
    console.error('Error al iniciar sesión:', error);
    alert('No se pudo iniciar sesión. Verifique sus credenciales e intente nuevamente.');
  }
};
```

- Beneficio: Los usuarios reciben retroalimentación clara y los errores no interrumpen la funcionalidad de la interfaz.

d. Registro de Errores (Logging)

- Descripción:
    - Todos los errores críticos se registran en un sistema centralizado para su análisis posterior.
- Implementación:
    - Uso de Winston para registrar errores en archivos y/o servicios en la nube como Sentry.

```
const winston = require('winston');


const logger = winston.createLogger({
  level: 'error',
  format: winston.format.json(),
```

```
  transports: [
    new winston.transports.File({ filename: 'logs/errors.log' }),
  ],
});


app.use((err, req, res, next) => {
  logger.error({ message: err.message, stack: err.stack });
  res.status(500).json({ error: 'Error interno del servidor' });
});
```

- Beneficio: Facilita la identificación y resolución de problemas en entornos de desarrollo y producción.

e. Manejo de Excepciones en la Base de Datos

- Descripción:
    - Los errores relacionados con consultas a la base de datos, como violaciones de claves únicas o problemas de conexión, se manejan de forma específica.
- Implementación:
    - Uso de capturas de errores (try-catch) al interactuar con la base de datos.

```
try {
  const usuario = await Usuario.create({ email, password });
  res.status(201).json(usuario);
```

```
} catch (error) {
  if (error.name === 'SequelizeUniqueConstraintError') {
    return res.status(400).json({ error: 'El correo ya está registrado' });
  }
  next(error); // Pasar errores no manejados al middleware global
}
```

- Beneficio: Los errores de base de datos se gestionan sin comprometer la estabilidad del sistema.

f. Respuestas de Error Consistentes

- Descripción:
    - Todos los errores se manejan con mensajes claros y estructurados para el cliente.
- Implementación:
    - Formato de respuesta estándar:

```
{
  "error": "Datos inválidos proporcionados",
  "details": "El campo 'email' es obligatorio."
}
```

- Beneficio: Mejora la experiencia del desarrollador frontend y del usuario final al proporcionar información útil sobre el error.

III. VIII. III. Beneficios del Manejo de Errores

1. Estabilidad del Sistema:
    - Los errores no provocan fallos críticos que puedan comprometer la funcionalidad del sistema.
2. Experiencia del Usuario:
    - Los usuarios reciben mensajes útiles y amigables, evitando frustraciones y confusiones.
3. Facilidad de Depuración:
    - Los desarrolladores cuentan con registros detallados para identificar y resolver problemas de manera eficiente.
4. Prevención de Problemas de Seguridad:
    - El sistema no expone información técnica sensible a los usuarios finales.

III. VIII. IV. Ejemplo Práctico del Flujo de Manejo de Errores

1. Un usuario intenta registrarse con un correo ya existente.
2. El backend detecta la violación de la clave única en la base de datos.
    - Se lanza una excepción SequelizeUniqueConstraintError.
3. El middleware de manejo de errores captura la excepción.
    - Se registra el error en un archivo de log.
    - Se envía una respuesta al cliente con un mensaje claro: "El correo ya está registrado."
4. El frontend muestra el mensaje de error al usuario.

III. IX. Documentación de los Componentes Desarrollados

La documentación de los componentes desarrollados es un pilar fundamental para garantizar la comprensión, mantenibilidad y escalabilidad del sistema. Una documentación bien estructurada proporciona a los desarrolladores actuales y futuros la información necesaria para comprender cómo funciona cada componente, cómo interactúa con otros módulos y cómo puede ser reutilizado o extendido. A continuación, se detalla cómo se abordó la documentación de los componentes en RegCon, incluyendo las herramientas utilizadas, la metodología seguida y los beneficios obtenidos.

III. IX. I. Objetivos de la Documentación

1. Facilitar el Entendimiento:
    - Proveer información clara y detallada sobre la funcionalidad, estructura y uso de cada componente.
2. Promover la Reutilización:
    - Permitir que los desarrolladores identifiquen componentes reutilizables y cómo integrarlos en nuevas funcionalidades.
3. Asegurar la Mantenibilidad:
    - Documentar la lógica interna y las dependencias de cada componente para facilitar la resolución de problemas y el refactorizado.
4. Reducir la Curva de Aprendizaje:
    - Ayudar a nuevos desarrolladores a integrarse rápidamente al equipo al ofrecer una referencia completa de los componentes.
5. Establecer Consistencia:

- Crear un estándar de documentación que pueda ser aplicado a todos los componentes del sistema.

III. IX. II. Metodología de Documentación

2. Metodología de Documentación

Para garantizar una documentación completa y uniforme, se adoptó una metodología basada en estándares de la industria, que incluye los siguientes pasos:

a. Identificación de Componentes Clave

- Todos los componentes desarrollados se clasificaron en categorías, como:
  - Componentes de presentación (UI).
  - Componentes funcionales.
  - Servicios y utilidades.

b. Uso de Comentarios en el Código

- Cada componente incluye comentarios descriptivos siguiendo el estándar JSDoc, lo que facilita la generación automática de documentación.

```
/**
 * Componente genérico para renderizar un botón.
 * @param {string} label - Etiqueta del botón.
 * @param {function} onClick - Función a ejecutar al hacer clic.
 * @param {string} type - Tipo del botón (e.g., "button", "submit").
 * @returns {JSX.Element} Componente Button.
 */
```

```
export const Button = ({ label, onClick, type = 'button' }) => {
  return (
    <button type={type} onClick={onClick}>
      {label}
    </button>
  );
};
```

c. Creación de Documentación Externa

- Además de los comentarios en el código, se generaron documentos externos que describen los componentes y su uso.
- Estos documentos incluyen:
    - Descripción general del componente.
    - Lista de propiedades (props) y sus tipos.
    - Ejemplos de uso.

d. Herramientas para Documentación Automática

- Se utilizó Storybook para documentar componentes visuales.
    - Permite visualizar y probar componentes de forma interactiva.
    - Ejemplo de configuración de Storybook para el componente Button:

```
export default {
  title: 'Componentes/Button',
```

Plan Final

```
  component: Button,

};

const Template = (args) => <Button {...args} />;

export const Primario = Template.bind({});

Primario.args = {

  label: 'Guardar',

  onClick: () => alert('Botón clickeado'),

  type: 'submit',

};
```

- JSDoc: Herramienta para generar documentación HTML basada en los comentarios del código.
- Markdown: Documentos manuales escritos en formato Markdown para incluir ejemplos y diagramas en el repositorio del proyecto.

III. IX. III. Estructura de la Documentación

3. Estructura de la Documentación

La documentación de cada componente sigue un formato estándar para garantizar claridad y uniformidad:

a. Nombre del Componente

- Ejemplo: Button

b. Descripción General

- Una breve descripción de lo que hace el componente.

- Ejemplo: "El componente Button es un botón genérico utilizado en toda la aplicación. Soporta diferentes estilos y tamaños mediante props."

c. Propiedades (Props)

- Lista de todas las propiedades que acepta el componente, incluyendo:
  - Nombre de la propiedad.
  - Tipo de dato.
  - Valor predeterminado (si aplica).
  - Descripción.

Propiedad	Tipo	Valor por Defecto	Descripción
`label`	`string`	`''`	Texto que se mostrará en el botón.
`onClick`	`function`	`null`	Función a ejecutar al hacer clic.
`type`	`string`	`'button'`	Tipo del botón (e.g., "submit").

d. Ejemplo de Uso

- Muestra cómo usar el componente en diferentes contextos.

```
<Button
  label="Guardar"
  onClick={() => console.log('Guardado')}
  type="submit"
/>
```

e. Dependencias

- Lista de cualquier librería o módulo que el componente necesita para funcionar.

- Ejemplo: "PropTypes"

f. Notas Adicionales

- Consideraciones importantes sobre el uso del componente, como:
  - "Este componente debe usarse junto con el tema global de TailwindCSS."

III. IX. IV. Documentación de Componentes

1. Navbar
   - Descripción: El componente Navbar es la barra superior de estado, es un componente crucial que muestra si se está actualizando o sincronizando a la base de datos y notificaciones.

```
import React from 'react';
import { Link } from 'react-router-dom';
import { FaBell, FaUser, FaCheck  } from 'react-icons/fa';
import { IoGrid } from "react-icons/io5";
import './Navbar.css';
import logo from '../../assets/regcon-logo.png'

export default function Navbar() {
    // Obtener la información del administrador actual desde localStorage
    const adminId = localStorage.getItem('user_id');
    const adminPicture = localStorage.getItem('admin_picture') || 'https://pbs.twimg.com/media/Gbmg1syWgAAcmGW?format=png&name=360x360'; // Imagen por defecto si no hay

    return (
        <nav className="custom-navbar fixed top-0 left-0 w-full z-50 shadow-md">
            <div className="max-w-screen-xl mx-auto px-4 py-3 flex justify-between items-center">
                <div className="flex items-center">
                    <img src={logo} className="custom-logo" alt="RegCon Logo" />
                </div>

                <div className="custom-icons flex items-center space-x-4">
                    <button className="notis hover:text-white">
                        <FaBell />
                    </button>
                    <button className="grid-c hover:text-white">
                        <FaCheck  />
                    </button>
                    <Link to={`/profile/${adminId}`}>
                        <img
                            src={adminPicture}
```

```
                            alt="User"
                            className="profile w-10 h-10 rounded-full border border-
    gray-600"
                        />
                    </Link>
                </div>
            </div>
        </nav>
    );
}

```

- Dependencias: React, React-router-dom, react-icons

2. Sidebar

- Descripción: El sidebar es otro componente importante, puesto que en este se muestran las diversas opciones que los administradores tienen para gestionar o visualizar los datos.

```
import React, { useState } from 'react';
import { Link } from 'react-router-dom';
import { FaHome, FaCheckCircle, FaRegFile, FaChartPie } from "react-icons/fa";
import { HiOutlineUserGroup } from "react-icons/hi";
import { MdEvent, MdCollectionsBookmark } from "react-icons/md";
import { IoMdSettings } from "react-icons/io";

export default function Sidebar() {
    const [isRegistroOpen, setIsRegistroOpen] = useState(false);
    const [isBoletosOpen, setIsBoletosOpen] = useState(false);
    const [isUsuariosOpen, setIsUsuariosOpen] = useState(false);
    const [isEventosOpen, setIsEventosOpen] = useState(false);
    const [isEquipoOpen, setIsEquipoOpen] = useState(false);

    const toggleRegistroMenu = () => {
        setIsRegistroOpen(!isRegistroOpen);
        setIsBoletosOpen(false);
        setIsUsuariosOpen(false);
        setIsEventosOpen(false);
        setIsEquipoOpen(false);
    };

    const toggleBoletosMenu = () => {
        setIsBoletosOpen(!isBoletosOpen);
```

```
        setIsRegistroOpen(false);
        setIsUsuariosOpen(false);
        setIsEventosOpen(false);
        setIsEquipoOpen(false);
    };

    const toggleUsuariosMenu = () => {
        setIsUsuariosOpen(!isUsuariosOpen);
        setIsRegistroOpen(false);
        setIsBoletosOpen(false);
        setIsEventosOpen(false);
        setIsEquipoOpen(false);
    };

    const toggleEventosMenu = () => {
        setIsEventosOpen(!isEventosOpen);
        setIsRegistroOpen(false);
        setIsBoletosOpen(false);
        setIsUsuariosOpen(false);
        setIsEquipoOpen(false);
    };

    const toggleEquipoMenu = () => {
        setIsEquipoOpen(!isEquipoOpen);
        setIsRegistroOpen(false);
        setIsBoletosOpen(false);
        setIsUsuariosOpen(false);
        setIsEventosOpen(false);
    };

    const adminId = localStorage.getItem('user_id'); // Obtener el ID del administrador

    return (
        <div>
            <aside
                id="separator-sidebar"
                className="fixed top-16 left-0 z-40 w-64 h-screen transition-transform -
translate-x-full sm:translate-x-0 bg-[#EB6D1E]"
                aria-label="Sidebar"
            >
                <div className="h-full px-3 py-4 overflow-y-auto bg-[#EB6D1E]">
                    <ul className="space-y-2 font-medium">
                        <li>
                            <Link
                                to="/dashboard"
                                className="flex items-center p-2 text-[#EDEDED] rounded-lg
hover:bg-orange-800 group"
                            >
                                <FaHome className="w-5 h-5" />
                                <span className="ms-3">Inicio</span>
                            </Link>
                        </li>
                        <li>
```

```
                            <Link to="/attendance-validation" className="flex items-center p-2
text-[#EDEDED] rounded-lg hover:bg-orange-800 group">
                                <FaCheckCircle className="w-5 h-5" />
                                <span className="ms-3">Validar Asistencia</span>
                            </Link>
                        </li>
                        <li>
                            <button
                                type="button"
                                className="flex items-center w-full p-2 text-base text-
[#EDEDED] transition duration-75 rounded-lg group hover:bg-orange-800"
                                onClick={toggleRegistroMenu}
                            >
                                <MdCollectionsBookmark className="w-5 h-5" />
                                <span className="flex-1 ms-3 text-left whitespace-
nowrap">Registro</span>
                                <svg className="w-3 h-3" aria-hidden="true"
xmlns="http://www.w3.org/2000/svg" fill="none" viewBox="0 0 10 6">
                                    <path stroke="currentColor" strokeLinecap="round"
strokeLinejoin="round" strokeWidth={2} d="m1 1 4 4 4-4" />
                                </svg>
                            </button>
                            <ul className={`py-2 space-y-2 ${isRegistroOpen ? 'block' :
'hidden'}`}>
                                <li>
                                    <Link
                                        to="/register"
                                        className="flex items-center w-full p-2 text-[#EDEDED]
transition duration-75 rounded-lg pl-11 group hover:bg-orange-800"
                                    >
                                        Consultar Registros
                                    </Link>
                                </li>
                                <li>
                                    <Link
                                        to="/register/add"
                                        className="flex items-center w-full p-2 text-[#EDEDED]
transition duration-75 rounded-lg pl-11 group hover:bg-orange-800"
                                    >
                                        Crear Registros
                                    </Link>
                                </li>

                                <li>
                                    <Link
                                        to="/register/print"
                                        className="flex items-center w-full p-2 text-[#EDEDED]
transition duration-75 rounded-lg pl-11 group hover:bg-orange-800"
                                    >
                                        Crear Informe
                                    </Link>
                                </li>
                            </ul>
                        </li>
```

```
                        <li>
                            <button
                                type="button"
                                className="flex items-center w-full p-2 text-base text-
[#EDEDED] transition duration-75 rounded-lg group hover:bg-orange-800"
                                onClick={toggleBoletosMenu}
                            >
                                <FaRegFile className="w-5 h-5" />
                                <span className="flex-1 ms-3 text-left whitespace-
nowrap">Boletos</span>
                                <svg className="w-3 h-3" aria-hidden="true"
xmlns="http://www.w3.org/2000/svg" fill="none" viewBox="0 0 10 6">
                                    <path stroke="currentColor" strokeLinecap="round"
strokeLinejoin="round" strokeWidth={2} d="m1 1 4 4 4-4" />
                                </svg>
                            </button>
                            <ul className={`py-2 space-y-2 ${isBoletosOpen ? 'block' :
'hidden'}`}>
                                <li>
                                    <Link
                                        to="/tickets"
                                        className="flex items-center w-full p-2 text-[#EDEDED]
transition duration-75 rounded-lg pl-11 group hover:bg-orange-800"
                                    >
                                        Administrar Boletos
                                    </Link>
                                </li>
                                <li>
                                    <Link
                                        to='/tickets/add'
                                        className="flex items-center w-full p-2 text-[#EDEDED]
transition duration-75 rounded-lg pl-11 group hover:bg-orange-800"
                                    >
                                        Crear Boletos
                                    </Link>
                                </li>

                                <li>
                                    <Link
                                        to="/tickets/print"
                                        className="flex items-center w-full p-2 text-[#EDEDED]
transition duration-75 rounded-lg pl-11 group hover:bg-orange-800"
                                    >
                                        Crear Informe
                                    </Link>
                                </li>
                            </ul>
                        </li>
                        <li>
                            <button
                                type="button"
                                className="flex items-center w-full p-2 text-base text-
[#EDEDED] transition duration-75 rounded-lg group hover:bg-orange-800"
                                onClick={toggleUsuariosMenu}
```

```
                            >
                                <HiOutlineUserGroup className="w-5 h-5" />
                                <span className="flex-1 ms-3 text-left whitespace-
nowrap">Usuarios</span>
                                <svg className="w-3 h-3" aria-hidden="true"
xmlns="http://www.w3.org/2000/svg" fill="none" viewBox="0 0 10 6">
                                    <path stroke="currentColor" strokeLinecap="round"
strokeLinejoin="round" strokeWidth={2} d="m1 1 4 4 4-4" />
                                </svg>
                            </button>
                            <ul className={`py-2 space-y-2 ${isUsuariosOpen ? 'block' :
'hidden'}`}>
                                <li>
                                    <Link
                                        to="/users"
                                        className="flex items-center w-full p-2 text-[#EDEDED]
transition duration-75 rounded-lg pl-11 group hover:bg-orange-800"
                                    >
                                        Administrar Usuarios
                                    </Link>
                                </li>
                                <li>
                                    <Link
                                        to="/users/add"
                                        className="flex items-center w-full p-2 text-[#EDEDED]
transition duration-75 rounded-lg pl-11 group hover:bg-orange-800"
                                    >
                                        Registrar Usuarios
                                    </Link>
                                </li>

                                <li>
                                    <Link
                                        to="/users/print"
                                        className="flex items-center w-full p-2 text-[#EDEDED]
transition duration-75 rounded-lg pl-11 group hover:bg-orange-800"
                                    >
                                        Crear Informe
                                    </Link>
                                </li>
                            </ul>
                        </li>
                        <li>
                            <button
                                type="button"
                                className="flex items-center w-full p-2 text-base text-
[#EDEDED] transition duration-75 rounded-lg group hover:bg-orange-800"
                                onClick={toggleEventosMenu}
                            >
                                <MdEvent className="w-5 h-5" />
                                <span className="flex-1 ms-3 text-left whitespace-
nowrap">Eventos</span>
                                <svg className="w-3 h-3" aria-hidden="true"
xmlns="http://www.w3.org/2000/svg" fill="none" viewBox="0 0 10 6">
```

```
                                    <path stroke="currentColor" strokeLinecap="round"
strokeLinejoin="round" strokeWidth={2} d="m1 1 4 4 4-4" />
                                </svg>
                            </button>
                            <ul className={`py-2 space-y-2 ${isEventosOpen ? 'block' :
'hidden'}`}>
                                <li>
                                    <Link
                                        to="/events"
                                        className="flex items-center w-full p-2 text-[#EDEDED]
transition duration-75 rounded-lg pl-11 group hover:bg-orange-800"
                                    >
                                        Administrar Eventos
                                    </Link>
                                </li>
                                <li>
                                    <Link
                                        to="/events/add"
                                        className="flex items-center w-full p-2 text-[#EDEDED]
transition duration-75 rounded-lg pl-11 group hover:bg-orange-800"
                                    >
                                        Crear Eventos
                                    </Link>
                                </li>
                            </ul>
                        </li>
                        <li>
                            <button
                                type="button"
                                className="flex items-center w-full p-2 text-base text-
[#EDEDED] transition duration-75 rounded-lg group hover:bg-orange-800"
                                onClick={toggleEquipoMenu}
                            >
                                <HiOutlineUserGroup className="w-5 h-5" />
                                <span className="flex-1 ms-3 text-left whitespace-
nowrap">Equipo</span>
                                <svg className="w-3 h-3" aria-hidden="true"
xmlns="http://www.w3.org/2000/svg" fill="none" viewBox="0 0 10 6">
                                    <path stroke="currentColor" strokeLinecap="round"
strokeLinejoin="round" strokeWidth={2} d="m1 1 4 4 4-4" />
                                </svg>
                            </button>
                            <ul className={`py-2 space-y-2 ${isEquipoOpen ? 'block' :
'hidden'}`}>
                                <li>
                                    <Link
                                        to="/my-workgroup"
                                        className="flex items-center w-full p-2 text-[#EDEDED]
transition duration-75 rounded-lg pl-11 group hover:bg-orange-800"
                                    >
                                        Mi Grupo de Trabajo
                                    </Link>
                                </li>
                                <li>
```

```
                                    <Link
                                        to={`/profile/${localStorage.getItem('user_id')}`} //
Cambiar a redirigir al perfil del administrador actual
                                        className="flex items-center w-full p-2 text-[#EDEDED]
transition duration-75 rounded-lg pl-11 group hover:bg-orange-800"
                                    >
                                        Mi Perfil
                                    </Link>
                                </li>
                            </ul>
                        </li>
                        <li>
                            <Link to="/stadistics" className="flex items-center p-2 text-
[#EDEDED] rounded-lg hover:bg-orange-800 group">
                                <FaChartPie className="w-5 h-5" />
                                <span className="ms-3">Estadísticas</span>
                            </Link>
                        </li>
                        <li>
                            <Link to="/settings" className="flex items-center p-2 text-
[#EDEDED] rounded-lg hover:bg-orange-800 group">
                                <IoMdSettings className="w-5 h-5" />
                                <span className="ms-3">Ajustes</span>
                            </Link>
                        </li>
                    </ul>
                </div>
            </aside>
        </div>
    );
}
```

- Dependencias: React, React-router-dom, react-icons

3. Login Form

- Descripción: El formulario de inicio de sesión también es crucial, puesto que sirve para que el administrador pueda entrar a la aplicación para poder interactuar con ella de forma directa.

```
import React, { useState } from 'react'; // Importar useState para manejar el estado
import './LoginForm.css';
import { useNavigate } from 'react-router-dom';

export default function LoginForm() {
    const navigate = useNavigate();
```

```
    const [email, setEmail] = useState(''); // Estado para el correo electrónico
    const [password, setPassword] = useState(''); // Estado para la contraseña
    const [error, setError] = useState(''); // Estado para manejar errores

    const handleSubmit = async (e) => {
        e.preventDefault();
        try {
            const response = await fetch('http://localhost:3000/login', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json',
                },
                body: JSON.stringify({ email, password }),
            });

            const data = await response.json();
            if (response.ok) {
                // Guardar el token, workgroup_id, user_id y role_id en localStorage
                localStorage.setItem('token', data.token);
                localStorage.setItem('workgroup_id', data.workgroup_id);
                localStorage.setItem('user_id', data.user_id); // Almacenar el ID del usuario
                localStorage.setItem('role_id', data.role_id); // Almacenar el role_id
                navigate('/dashboard'); // Navega a la página de dashboard o donde desees
            } else {
                setError(data.error);
            }
        } catch (error) {
            console.error('Error en la solicitud de inicio de sesión:', error);
            setError('Error al intentar iniciar sesión.');
        }
    };

    return (
        <div className='main-container'>
            <div className='custom-form'>
                <form className="max-w-sm mx-auto" onSubmit={handleSubmit}>
                    <div className="info-form">
                        <h1 className='title-form'>Iniciar Sesión</h1>
                        <p className='description-form'>
                            Inicia sesión con las credenciales con las que te registraste en
RegCon™ o las proporcionadas por tu administrador
                        </p>
                    </div>
                    {error && <p className="text-red-500">{error}</p>} {/* Mostrar mensaje de
error si hay */}
                    <div className="mb-5">
                        <label htmlFor="email" className="block mb-2 text-sm font-medium text-
gray-900">Correo</label>
                        <input
                            type="email"
                            id="email"
                            value={email} // Vincular el valor del input al estado
                            onChange={(e) => setEmail(e.target.value)} // Actualizar el estado
```

```
                            className="border text-sm rounded-lg focus:ring-blue-500
focus:border-blue-500 block w-full p-2.5"
                            placeholder="admin@mymail.com"
                            required=""
                        />
                    </div>
                    <div className="mb-5">
                        <label htmlFor="password" className="block mb-2 text-sm font-medium
text-gray-900">Contraseña</label>
                        <input
                            type="password"
                            id="password"
                            value={password} // Vincular el valor del input al estado
                            onChange={(e) => setPassword(e.target.value)} // Actualizar el
estado
                            className="border rounded-lg focus:ring-blue-500 focus:border-
blue-500 block w-full p-2.5"
                            required=""
                        />
                    </div>
                    <div className='button-to-access'>
                        <button
                            type="submit"
                            className="text-white bg-blue-700 hover:bg-blue-800 focus:ring-4
focus:outline-none focus:ring-blue-300 font-medium rounded-lg text-sm w-full sm:w-auto px-5
py-2.5 text-center dark:bg-blue-600 dark:hover:bg-blue-700 dark:focus:ring-blue-800"
                        >
                            Acceder
                        </button>
                    </div>
                    <p className='pics-as mt-2'>¿No tienes una cuenta? <a className="just-it-
a" href="">Regístrate aquí</a></p>
                </form>
            </div>
        </div>
    );
}
```

4. Validate Attendance Form

- Descripción: Este componente permite validar si un asistente tiene su presencia dentro del evento al que asistió y permite registrar su asistencia.

```
import React, { useState } from 'react';
import QRCodeScannerModal from '../qrScannerModal/QRCodeScannerModal';
import ErrorModal from '../errorModal/ErrorModal';
import './ValidateAttendanceForm.css';
```

```
export default function ValidateAttendanceForm() {
    const [ticketCode, setTicketCode] = useState('');
    const [ticketInfo, setTicketInfo] = useState(null);
    const [error, setError] = useState('');
    const [errorModalVisible, setErrorModalVisible] = useState(false);
    const [modalMessage, setModalMessage] = useState('');
    const [qrModalVisible, setQrModalVisible] = useState(false);

    const handleInputChange = (e) => {
        setTicketCode(e.target.value);
    };

    const handleValidate = async () => {
        if (!ticketCode.trim()) {
            setError('El código del boleto no puede estar vacío.');
            return;
        }

        try {
            const workgroupId = localStorage.getItem('workgroup_id'); // Obtener el
workgroup_id de la sesión
            const response = await fetch(`http://localhost:3000/ticket-view-
code/${ticketCode}?workgroup_id=${workgroupId}`);
            const data = await response.json();

            if (response.ok) {
                setTicketInfo(data.data);
                await validateAttendance(data.data); // Validar la asistencia después de
obtener información del boleto
            } else {
                setError(data.message || 'Este boleto no existe o es inválido.');
                setTicketInfo(null);
                setModalMessage('No se encontró registro de asistencia para este boleto.');
                setErrorModalVisible(true);
            }
        } catch (error) {
            console.error('Error:', error);
            setModalMessage('Error al validar el boleto.');
            setErrorModalVisible(true);
        }
    };

    const validateAttendance = async (ticketData) => {
        try {
            const attendanceResponse = await
fetch(`http://localhost:3000/attendance?ticket_code=${ticketData.code}&workgroup_id=${ticketDa
ta.workgroup_id}`);
            const attendanceData = await attendanceResponse.json();

            if (attendanceResponse.ok && attendanceData.data.length > 0) {
                const attendanceRecord = attendanceData.data[0];

                if (attendanceRecord.status === 'Sin Asistencia') {
                    // Actualizar estado a 'Ha Asistido'
```

```
                    await updateAttendance(attendanceRecord.id);
                } else {
                    setModalMessage('Este usuario ya ha sido validado.');
                    setErrorModalVisible(true);
                }
            } else {
                setModalMessage('No se encontró registro de asistencia para este boleto.');
                setErrorModalVisible(true);
            }
        } catch (error) {
            console.error('Error al validar asistencia:', error);
            setModalMessage('Error al validar la asistencia.');
            setErrorModalVisible(true);
        }
    };

    const updateAttendance = async (attendanceId) => {
        try {
            const response = await fetch(`http://localhost:3000/attendance-
status/${attendanceId}`, {
                method: 'PUT',
                headers: {
                    'Content-Type': 'application/json'
                },
                body: JSON.stringify({ status: 'Ha Asistido' }) // Solo cambiar el estado
            });

            if (response.ok) {
                alert('Asistencia registrada correctamente');
                // Aquí puedes reiniciar los datos del formulario si es necesario
            } else {
                setModalMessage('Error al actualizar el estado de asistencia.');
                setErrorModalVisible(true);
            }
        } catch (error) {
            console.error('Error:', error);
            setModalMessage('Error al actualizar la asistencia');
            setErrorModalVisible(true);
        }
    };

    return (
        <div className="custom-cs-ab validate-attendance-form p-4 bg-white rounded-lg shadow-
md">
            <h2 className="title-uts-avs">Registrar Asistencia</h2>
            <div className="mb-4">
                <label htmlFor="ticketCode" className="block text-sm font-medium text-gray-
700">Código del Boleto</label>
                <input
                    type="text"
                    id="ticketCode"
                    value={ticketCode}
                    onChange={handleInputChange}
                    className="mt-1 block w-full border rounded-md p-2"
```

```
                />
                <button
                    type="button"
                    onClick={() => setQrModalVisible(true)}
                    className='bg-yellow-400 py-1 px-2 mt-1 rounded font-medium hover:bg-
yellow-600'
                >
                    Escanear por QR
                </button>
            </div>
            <button
                onClick={handleValidate}
                className="bg-teal-400 text-white py-2 px-4 rounded hover:bg-teal-500"
            >
                Validar Boleto
            </button>

            {/* Información del ticket */}
            {ticketInfo && (
                <div className="mt-4 p-4 border rounded bg-green-100">
                    <h3 className="text-lg font-bold">Información del Boleto</h3>
                    <p><strong>Código:</strong> {ticketInfo.code}</p>
                    <p><strong>Nombre:</strong> {ticketInfo.ticket_name}</p>
                    <p><strong>Categoría:</strong> {ticketInfo.category_name}</p>
                    <p><strong>Costo:</strong> ${ticketInfo.category_price}</p>
                    <p><strong>Descripción:</strong> {ticketInfo.category_description}</p>
                    <p><strong>Estado:</strong> {ticketInfo.status}</p>
                </div>
            )}

            {/* Modal para mensajes de error */}
            {errorModalVisible && (
                <ErrorModal
                    message={modalMessage}
                    onClose={() => setErrorModalVisible(false)}
                />
            )}

            {/* Modal de Escaneo QR */}
            {qrModalVisible && (
                <QRCodeScannerModal
                    onClose={() => setQrModalVisible(false)}
                    onCodeDetected={setTicketCode}
                />
            )}
        </div>
    );
}
```

5. Ticket Validation Form

- Descripción: Este componente permite validar cuando un boleto tiene validez y si es funcional, para que pueda registrarse su uso y cambiar su estado de Disponible a Usado.

```
import React, { useState } from 'react';
import { useNavigate } from 'react-router-dom';
import './TicketValidationForm.css';

export default function TicketValidationForm() {
    const navigate = useNavigate();
    const [ticketCode, setTicketCode] = useState('');
    const [ticketInfo, setTicketInfo] = useState(null);
    const [error, setError] = useState('');

    const handleInputChange = (e) => {
        setTicketCode(e.target.value);
    };

    const handleValidate = async () => {
        if (!ticketCode.trim()) {
            setError('El código del boleto no puede estar vacío.'); // Error si el campo está vacío
            setTicketInfo(null);
            return;
        }

        console.log("Validando código del boleto:", ticketCode); // Agregado para depuración

        try {
            const workgroupId = localStorage.getItem('workgroup_id'); // Obtener el workgroup_id de la sesión
            const response = await fetch(`http://localhost:3000/ticket-view/${ticketCode}?workgroup_id=${workgroupId}`); // Enviar workgroup_id
            const data = await response.json();
            console.log("Respuesta de la API:", data); // Agregado para depuración
            if (response.ok) {
                setTicketInfo(data.data);
                setError('');
            } else {
                setError(data.message || 'Este boleto no existe o es inválido.'); // Mensaje de error más específico
                setTicketInfo(null);
            }
        } catch (error) {
            console.error('Error:', error);
            setError('Error al validar el boleto.');
            setTicketInfo(null);
        }
    };

    const handleAddCategory = () => {
```

```
        navigate('/tickets/validate/qr'); // Navegar a la página de escaneo QR
    };

    return (
        <div className="custom-cs-tb-tb p-4 bg-white rounded-lg shadow-md">
            <div className="flex justify-between items-center mb-4">
                <h2 className="title-uts">Validar Boletos</h2>
                <button
                    className="bg-teal-400 text-white py-2 px-4 rounded hover:bg-teal-500"
                    onClick={handleAddCategory}
                >
                    Escaneo por QR
                </button>
            </div>
            <div className="mb-4">
                <label htmlFor="ticketCode" className="block text-sm font-medium text-gray-
700">Código del Boleto</label>
                <input
                    type="text"
                    id="ticketCode"
                    value={ticketCode}
                    onChange={handleInputChange}
                    className="mt-1 block w-full border rounded-md p-2"
                    placeholder="Escribe el código del boleto"
                />
            </div>
            <button
                onClick={handleValidate}
                className="bg-teal-400 text-white py-2 px-4 rounded hover:bg-teal-500"
            >
                Validar Boleto
            </button>

            {/* Mensaje de error */}
            {error && (
                <div className="mt-4 p-4 border rounded bg-red-100">
                    <p className="text-red-500">{error}</p>
                </div>
            )}

            {/* Información del ticket */}
            {ticketInfo && (
                <div className="mt-4 p-4 border rounded bg-green-100">
                    <h3 className="text-lg font-bold">Información del Boleto</h3>
                    <p><strong>Código:</strong> {ticketInfo.code}</p>
                    <p><strong>Nombre:</strong> {ticketInfo.ticket_name}</p>
                    <p><strong>Categoría:</strong> {ticketInfo.category_name}</p>
                    <p><strong>Costo:</strong> ${ticketInfo.category_price}</p>
                    <p><strong>Descripción:</strong> {ticketInfo.category_description}</p>
                    <p><strong>Estado:</strong> {ticketInfo.status}</p>
                </div>
            )}
        </div>
    );
```

```
}
```

6. QR Code Generator
    - Este componente permite generar los QR necesarios para los boletos y registros de asistencia

```
import React from 'react';
import { FaFileDownload } from "react-icons/fa";

const QrCodeGenerator = ({ code }) => {
    const qrCodeUrl = `https://api.qrserver.com/v1/create-qr-code/?data=${code}&size=200x200`;

    const handleDownload = () => {
        // Create an anchor element
        const link = document.createElement('a');
        link.href = qrCodeUrl; // Set the URL for the image
        link.download = `QRCode-${code}.png`; // Set the name for the downloaded file
        document.body.appendChild(link); // Append the link to the body
        link.click(); // Simulate a click on the link to trigger the download
        document.body.removeChild(link); // Remove the link from the DOM
    };

    return (
        <div>
            <img src={qrCodeUrl} alt={`Ha ocurrido un error, intetelo de nuevo`} />
            <button className="download-button" onClick={handleDownload}>
            <FaFileDownload />
            </button>
        </div>
    );
};

export default QrCodeGenerator;
```

7. Event Stadistics
    - Este componente permite el poder visualizar información acerca de asistentes registrados, no asistidos y asistidos a los diversos eventos creados por el grupo de trabajo.

```
import React, { useEffect, useState } from 'react';
import { Bar } from 'react-chartjs-2';
```

```
import { Chart as ChartJS, Title, Tooltip, Legend, BarElement, CategoryScale, LinearScale }
from 'chart.js';
import './EventStatistics.css';

ChartJS.register(Title, Tooltip, Legend, BarElement, CategoryScale, LinearScale);

const EventStatistics = () => {
    const [eventData, setEventData] = useState([]);
    const [error, setError] = useState('');

    useEffect(() => {
        const fetchEventStatistics = async () => {
            const workgroupId = localStorage.getItem('workgroup_id');
            try {
                const response = await
fetch(`http://localhost:3000/eventattendancesummary?workgroup_id=${workgroupId}`);
                const data = await response.json();
                if (response.ok) {
                    setEventData(data.data);
                } else {
                    setError(data.message || 'Error al obtener estadísticas de eventos');
                }
            } catch (error) {
                console.error('Error:', error);
                setError('Error al obtener estadísticas de eventos');
            }
        };
        fetchEventStatistics();
    }, []);

    const chartData = {
        labels: eventData.map(event => event.event_name),
        datasets: [
            {
                label: 'Total Registrados',
                data: eventData.map(event => event.total_registered),
                backgroundColor: 'rgba(75, 192, 192, 0.2)',
                borderColor: 'rgba(75, 192, 192, 1)',
                borderWidth: 1,
            },
            {
                label: 'Total Asistidos',
                data: eventData.map(event => event.total_attended),
                backgroundColor: 'rgba(153, 102, 255, 0.2)',
                borderColor: 'rgba(153, 102, 255, 1)',
                borderWidth: 1,
            },
            {
                label: 'Total No Asistidos',
                data: eventData.map(event => event.total_not_attended),
                backgroundColor: 'rgba(255, 99, 132, 0.2)',
                borderColor: 'rgba(255, 99, 132, 1)',
                borderWidth: 1,
            },
```

```
        ],
    };

    return (
        <div className="custom-estats p-4 bg-white rounded-lg shadow-md">
            <h2 className="title-uts text-2xl font-bold mb-4">Estadísticas de Eventos</h2>
            {error && <p className="text-red-500">{error}</p>}
            {eventData.length > 0 ? (
                <div style={{ position: 'relative', height: '300px' }}>
                    <Bar
                        data={chartData}
                        options={{
                            responsive: true,
                            maintainAspectRatio: false,
                            scales: {
                                y: {
                                    beginAtZero: true,
                                    ticks: {
                                        autoSkip: true,
                                        maxTicksLimit: 5,
                                    }
                                }
                            }
                        }}
                    />
                </div>
            ) : (
                <p>Cargando estadísticas de eventos...</p>
            )}
        </div>
    );
};

export default EventStatistics;
```

8. App
    - Es la aplicación que funciona como router, para redirigir las peticiones a las páginas de acuerdo a donde el usuario desee ir.

```
import React from 'react';
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
import LogInPage from './pages/LogInPage';
import ScrollToTop from './components/scrollToTop/ScrollToTop';
import DashboardHome from './pages/DashboardHome';
import DashboardSettings from './pages/DashboardSettings';
import DashAddRegister from './pages/DashAddRegister';
import DashAddUsers from './pages/DashAddUsers';
import DashAdmUsers from './pages/DashAdmUsers';
```

```
import UsersEditUser from './pages/UsersEditUser';
import UsersPrint from './pages/UsersPrint';
import DashAddEvent from './pages/DashAddEvent';
import DashAdmEvents from './pages/DashAdmEvents';
import EventsEditEvent from './pages/EventsEditEvent';
import DashAddTickets from './pages/DashAddTickets';
import DashAdmTC from './pages/DashAdmTC';
import DashAddTC from './pages/DashAddTC';
import TCsEditTC from './pages/TCsEditTC';
import DashAdmTickets from './pages/DashAdmTickets';
import TicketCategorizedTableW from './pages/TicketCategorizedTableW';
import TicketValidation from './pages/TicketValidation';
import TicketValQR from './pages/TicketValQR';
import TicketsPrint from './pages/TicketsPrint';
import DashAdmRegister from './pages/DashAdmRegister';
import RegEditReg from './pages/RegEditReg';
import RegisterPrint from './pages/RegisterPrint';
import AttenVal from './pages/AttenVal';
import NotFound from './components/notFound/NotFound';
import MyWorkGroup from './pages/MyWorkGroup';
import ProfilePage from './pages/ProfilePage';
import Stadistic from './pages/Stadistic';
import NoWebCont from './pages/NoWebCont';

function App() {
  return (
    <Router>
      <ScrollToTop />
      <Routes>
        <Route path="/" element={<DashboardHome />} />
        <Route path="/login" element={<LogInPage />} />
        <Route path="/dashboard" element={<DashboardHome />} />
        <Route path="/settings" element={<DashboardSettings />} />
        <Route path="/users/add" element={<DashAddUsers />} />
        <Route path="/register/add" element={<DashAddRegister />} />
        <Route path="/users" element={<DashAdmUsers />} />
        <Route path="/users/edit/:id" element={<UsersEditUser />} />
        <Route path="/users/print" element={<UsersPrint />} />
        <Route path="/events/add" element={<DashAddEvent />} />
        <Route path="/events" element={<DashAdmEvents />} />
        <Route path="/events/edit/:id" element={<EventsEditEvent />} />
        <Route path="/tickets/add" element={<DashAddTickets />} />
        <Route path="/ticket-categories" element={<DashAdmTC />} />
        <Route path="/ticket-categories/add" element={<DashAddTC />} />
        <Route path="/ticket-categories/edit/:id" element={<TCsEditTC />} />
        <Route path="/tickets" element={<DashAdmTickets />} />
        <Route path="/tickets/categories/:category_id" element={<TicketCategorizedTableW />}
/>
        <Route path="/tickets/validate" element={<TicketValidation />} />
        <Route path="/tickets/validate/qr" element={<TicketValQR />} />
        <Route path="/tickets/print" element={<TicketsPrint />} />
        <Route path="/register" element={<DashAdmRegister />} />
        <Route path="/register/edit/:id" element={<RegEditReg />} />
        <Route path="/register/print" element={<RegisterPrint />} />
```

```
        <Route path="/attendance-validation" element={<AttenVal />} />
        <Route path="/my-workgroup" element={<MyWorkGroup />} />
        <Route path="/profile/:id" element={<ProfilePage />} />
        <Route path="/stadistics" element={<Stadistic />} />
        <Route path="/app/configuration" element={<NoWebCont />} /> //CAMBIAR RUTA A FUTURO
        <Route path="/app/database" element={<NoWebCont />} /> //CAMBIAR RUTA A FUTURO
        <Route path="*" element={<NotFound />} />
      </Routes>
    </Router>
  );
}

export default App;
```

III. IX. V. Beneficios de la Documentación

1. Facilita la Reutilización:
    - Los desarrolladores pueden identificar y utilizar componentes existentes en nuevas funcionalidades, reduciendo el tiempo de desarrollo.
2. Aumenta la Productividad:
    - La documentación clara y accesible reduce el tiempo necesario para comprender cómo funciona un componente.
3. Promueve la Mantenibilidad:
    - Los desarrolladores pueden modificar o mejorar componentes sin temor a romper funcionalidades existentes, gracias a la claridad de la documentación.
4. Reduce la Curva de Aprendizaje:
    - Los nuevos desarrolladores pueden integrarse rápidamente al proyecto al tener acceso a una referencia completa.
5. Establece Consistencia:

- La documentación uniforme asegura que todos los componentes sigan los mismos estándares de diseño y uso.

IV. Aseguramiento de Calidad

IV. I. Planificación de Pruebas

Este documento tiene como objetivo definir el plan de pruebas para asegurar que el sistema RegCon cumpla con los requisitos establecidos y funcione de acuerdo con las expectativas del usuario. Se probarán los componentes del sistema para detectar posibles errores, mejorar la calidad y asegurar una experiencia de usuario sin fallos.

IV. I. I. Objetivo de las Pruebas

- Validar que el registro de usuarios y la gestión de eventos funcionen correctamente.
- Comprobar que las funciones de compra de entradas y validación de asistencia estén integradas y sean fiables.
- Verificar la seguridad del sistema, especialmente en la autenticación de usuarios y la protección de datos.
- Asegurar que la interfaz de usuario sea amigable y responda adecuadamente en diferentes dispositivos.

IV. I. II. Alcance de las pruebas

Las pruebas se centrarán en las siguientes áreas:

- Registro de Usuarios: Verificar la creación y edición de usuarios en el sistema.
- Gestión de Eventos: Probar la creación, edición y eliminación de eventos, así como la asociación de eventos con categorías de tickets.
- Compra de Tickets: Comprobar que el proceso de compra de entradas se realice correctamente.

- Validación de Asistencia: Asegurar que el sistema registre correctamente la asistencia de los usuarios a eventos.
- Autenticación y Seguridad: Comprobar que los administradores y usuarios se autentiquen correctamente y que los datos estén protegidos.

IV. I. III. Estrategia de pruebas

IV. I. III. I. Tipos de Pruebas

- Pruebas Funcionales: Se verificará que todas las funcionalidades (registro, compra de entradas, gestión de eventos) estén operativas.
- Pruebas de Seguridad: Se realizarán pruebas de penetración y validación de la autenticación y autorización de usuarios.
- Pruebas de Usabilidad: Validar que la interfaz de usuario sea fácil de usar y accesible.
- Pruebas de Rendimiento: Evaluar la capacidad de respuesta del sistema durante la carga máxima de usuarios y eventos.
- Pruebas de Integración: Asegurar que los diferentes módulos del sistema (base de datos, servidor, frontend) interactúan correctamente.

IV. I. III. II. Herramientas de Pruebas

- Selenium: Para automatizar pruebas de la interfaz de usuario.
- Jest + React Testing Library: Para probar componentes de React.
- Postman: Para probar los endpoints de la API.
- OWASP ZAP: Para pruebas de seguridad.
- JMeter: Para pruebas de rendimiento.

IV. I. IV. Cronograma de Pruebas

Las pruebas se realizarán en varias fases:

- Fase 1: Planificación y Diseño (1 semana): Definir los escenarios de prueba y las herramientas.
- Fase 2: Ejecución de Pruebas Funcionales (2 semanas): Realizar pruebas unitarias y de integración.
- Fase 3: Pruebas de Seguridad y Usabilidad (1 semana): Evaluar la seguridad y la experiencia de usuario.
- Fase 4: Pruebas de Rendimiento (1 semana): Evaluar el sistema bajo carga.
- Fase 5: Revisión y Ajustes (1 semana): Analizar los resultados y corregir los problemas encontrados.

IV. I. V. Criterios de Aceptación

- El sistema debe pasar todas las pruebas funcionales con al menos un 90% de cobertura de código.
- La seguridad del sistema debe ser probada con herramientas automatizadas para detectar vulnerabilidades comunes.
- El sistema debe ser accesible y usable en diferentes dispositivos (desktop y móvil).
- El sistema debe ser capaz de manejar al menos 500 usuarios simultáneamente sin caídas de rendimiento.

IV. I. VI. Recursos Necesarios

- Equipo de pruebas: Un equipo de 3 personas: 1 líder de pruebas, 1 tester funcional y 1 tester de seguridad.
- Accesos: Acceso a la base de datos, API, y entorno de desarrollo para pruebas.

IV. I. VII. Riesgos y Mitigación

- Falta de tiempo para realizar pruebas exhaustivas: Realizar pruebas prioritarias según los riesgos del sistema.
- Problemas de configuración en el entorno de pruebas: Tener entornos de prueba duplicados listos para ser utilizados.
- Falta de cobertura en dispositivos móviles: Asegurarse de realizar pruebas de usabilidad en los principales navegadores y dispositivos.

IV. II. Preparación de Pruebas

Este documento de Preparación de Pruebas tiene como objetivo definir los detalles de las pruebas que se llevarán a cabo para garantizar que el sistema RegCon, el cual gestiona registros y asistencia a convenciones, funcione correctamente en todos los aspectos esenciales. El enfoque se centrará en pruebas funcionales, de rendimiento, de seguridad y de usabilidad, con el fin de asegurar una experiencia óptima para los usuarios finales.

IV. II. I. Objetivos del sistema RegCon

El sistema RegCon permite a los administradores gestionar convenciones, registros de usuarios, compra de entradas y control de asistencia a eventos. Las pruebas deben cubrir las siguientes funcionalidades clave:

- Registro de usuarios y gestión de perfiles.
- Creación y gestión de eventos, incluyendo la asignación de entradas.
- Validación de compra de entradas y generación de tickets.
- Validación de asistencia a los eventos.
- Seguridad, autenticación y protección de datos.
- Interfaz de usuario para facilitar la experiencia del usuario.

IV. II. II. Alcance de las pruebas

IV. II. II. I Áreas que se probarán:

- Funcionalidad de Registro:
    - Registro de nuevos usuarios.
    - Gestión de perfiles de usuario (actualización, eliminación).
    - Validación de correos electrónicos.
- Gestión de Eventos:
    - Creación de eventos, incluyendo definición de nombre, fecha, ubicación y descripción.
    - Asociar categorías de entradas a los eventos.
    - Modificación y eliminación de eventos.
- Compra y Validación de Tickets:
    - Realización de compras de entradas.
    - Generación de códigos QR para validación.
    - Envío de tickets a los usuarios registrados.
- Asistencia a Eventos:
    - Verificación de la asistencia de los usuarios mediante escaneo de QR.
    - Registro de las entradas validadas.
- Autenticación y Seguridad:
    - Pruebas de inicio de sesión para usuarios y administradores.

  - Autorización de roles (usuario regular vs. administrador).
  - Protección de datos sensibles, como contraseñas y transacciones.

- Interfaz de Usuario:
  - Usabilidad y accesibilidad en diferentes dispositivos (web y móvil).
  - Respuesta ante entradas no válidas o acciones incorrectas.

IV. II. II. II Áreas que no se probarán:

- Integración con sistemas de pago de terceros, como gateways (esto debe ser probado en un entorno de integración separado).
- Funcionalidades de administración de bases de datos (realizadas por el equipo de backend).

IV. II. III. Tipos de Pruebas

IV. II. III. I. Pruebas Funcionales:

- Pruebas de Caja Negra: Verificación de los requisitos del sistema sin tener en cuenta la implementación interna.
- Pruebas Unitarias: Se probarán las funciones básicas del sistema (registro de usuarios, compra de entradas, etc.).
- Pruebas de Integración: Se validará la interacción entre los módulos, como la base de datos con la interfaz y el servidor con las APIs.

IV. II. III. II Pruebas de Seguridad:

- Pruebas de Autenticación: Asegurarse de que solo los usuarios autorizados puedan acceder a las áreas del sistema.

- Pruebas de Autorización: Validar que los administradores tengan permisos adecuados y que los usuarios no accedan a información restringida.
- Pruebas de Vulnerabilidad: Uso de herramientas como OWASP ZAP para detectar vulnerabilidades comunes (inyección SQL, XSS, CSRF, etc.).

IV. II. III. III. Pruebas de Usabilidad:

- Evaluar la facilidad de uso del sistema por parte de los usuarios finales.
- Verificar la adecuación de la interfaz para dispositivos móviles y desktop.

IV. II. III. IV. Pruebas de Rendimiento:

- Pruebas de Carga: Evaluar cómo responde el sistema bajo una carga de 100-500 usuarios simultáneos.
- Pruebas de Estrés: Evaluar el comportamiento del sistema bajo una carga extrema (más de 500 usuarios simultáneos).

IV. II. III. V. Pruebas de Compatibilidad:

- Probar el sistema en diferentes navegadores web y dispositivos móviles para asegurar la compatibilidad (Chrome, Firefox, Safari, dispositivos iOS y Android).

IV. II. IV. Criterios de Entrada y de Salida

IV. II. IV. I Criterios de Entrada:

- El sistema debe estar completamente desarrollado y listo para pruebas (con todas las funcionalidades básicas implementadas).
- Todos los entornos de prueba deben estar configurados (servidores de backend, bases de datos, aplicaciones móviles, etc.).
- El equipo de desarrollo debe haber realizado pruebas unitarias y de integración previas.

IV. II. IV. II. Criterios de Salida:

- Todas las pruebas funcionales deben haber sido ejecutadas y los resultados deben cumplir los criterios establecidos.
- No deben quedar fallos críticos sin resolver (errores de seguridad, fallos en la compra de entradas, etc.).
- El sistema debe haber sido validado en todos los navegadores y dispositivos especificados.

IV. II. V. Recursos Necesarios

IV. II. V. I. Equipo de Pruebas:

- Líder de Pruebas: Responsable de coordinar todas las actividades de prueba y garantizar el cumplimiento de los plazos.
- Tester Funcional: Se encargará de ejecutar las pruebas de las funcionalidades del sistema (registro, compra de entradas, etc.).
- Tester de Seguridad: Responsable de realizar pruebas de penetración y validación de seguridad.
- Tester de Usabilidad: Se centrará en la experiencia del usuario y la interfaz.
- Tester de Rendimiento: Evaluará el rendimiento del sistema bajo diversas condiciones de carga.

IV. II. V. II. Herramientas de Pruebas:

- Jest + React Testing Library: Para las pruebas de componentes de React.
- Postman: Para probar los endpoints de la API.
- OWASP ZAP: Para realizar pruebas de seguridad.

- Selenium: Para automatizar las pruebas de interfaz.
- JMeter: Para realizar pruebas de rendimiento.

IV. II. V. III. Entornos de Pruebas:

- Entorno de Desarrollo: Usado por los desarrolladores para integrar y probar cambios.
- Entorno de Pruebas: Una réplica del entorno de producción donde se ejecutarán todas las pruebas.
- Entorno de Producción (para pruebas finales): Solo utilizado para pruebas de validación y rendimiento bajo condiciones reales.

IV. II. VI. Plan de Ejecución de Pruebas

IV. II. VI. I. Fase de Planificación:

- Duración: 1 semana.
- Objetivos: Definir los casos de prueba, asignar recursos y preparar los entornos de prueba.
- Entregables: Plan de pruebas finalizado, escenarios de prueba detallados.

IV. II. VI. II. Fase de Ejecución de Pruebas Funcionales:

- Duración: 2 semanas.
- Objetivos: Realizar pruebas de todas las funcionalidades clave (registro, compra de entradas, etc.).
- Entregables: Informe de errores y sugerencias de mejoras.

IV. II. VI. III. Fase de Pruebas de Seguridad:

- Duración: 1 semana.

- Objetivos: Evaluar la seguridad del sistema mediante pruebas de penetración y vulnerabilidad.
- Entregables: Informe de seguridad con vulnerabilidades encontradas.

IV. II. VI. IV. Fase de Pruebas de Usabilidad:

- Duración: 1 semana.
- Objetivos: Validar la experiencia de usuario en dispositivos móviles y escritorio.
- Entregables: Informe de usabilidad con recomendaciones.

IV. II. VI. V. Fase de Pruebas de Rendimiento:

- Duración: 1 semana.
- Objetivos: Evaluar el rendimiento del sistema bajo carga.
- Entregables: Informe de pruebas de carga y estrés.

IV. II. VII. Gestión de Incidencias

Todas las incidencias encontradas durante las pruebas se documentarán detalladamente en un sistema de seguimiento de errores, como JIRA, y se priorizarán según su impacto en el sistema (crítico, mayor, menor). Las incidencias deben ser corregidas antes de la siguiente fase de pruebas.

IV. II. VIII. Diseño de Casos de Pruebas

Tipo de caso de prueba	Descripción	Pasos	Resultado Esperado	Estado
Funcionalidad	Verificar registro de nuevos usuarios	Crear un nuevo usuario con todos los	El nuevo usuario se crea correctamente y	Pasó

		campos obligatorios llenados	se envía un correo de confirmación	
Funcionalidad	El nuevo usuario se crea correctamente y se envía un correo de confirmación	Realizar una compra de entrada para un evento disponible	La compra se procesa correctamente y el ticket se genera con los detalles correspondientes	Pasó
Seguridad	Verificar la autenticación de usuario	Intentar iniciar sesión con credenciales incorrectas	El sistema no debe permitir el acceso y mostrar un mensaje de error de autenticación	Pasó
Seguridad	Verificar protección de contraseñas	Crear una nueva contraseña que no cumpla con los requisitos de seguridad (mínimo 8 caracteres, mezcla de letras y números)	El sistema debe rechazar la contraseña y solicitar que se cumplan los requisitos	Pasó

Usabilidad	Verificar la interfaz de usuario en dispositivos móviles	Acceder al sistema desde un dispositivo móvil y navegar por las diferentes secciones	El diseño debe adaptarse correctamente, mostrando toda la información y sin errores visuales	Falló
Usabilidad	Validar facilidad para comprar entradas	Ingresar a la página de eventos y seleccionar una entrada para comprar	La opción de compra debe ser clara y fácil de usar, sin confusión al completar la transacción	Pasó
Funcionalidad	Verificar la creación y edición de eventos	Crear un evento nuevo, asignar categorías de entradas y editar la información del evento	El evento debe crearse correctamente y ser editable posteriormente	Pasó
Funcionalidad	Validar la escaneada de QR para la validación de asistencia	Escanear un código QR de un ticket comprado en el sistema	El sistema debe registrar la asistencia correctamente, y mostrar una	Pasó

			confirmación de validación	
Seguridad	Verificar autorización de roles (Administrador vs. Usuario regular)	Iniciar sesión como usuario regular y como administrador	El administrador debe tener acceso completo a las funciones de gestión, mientras que el usuario regular tiene acceso limitado	Pasó
Rendimiento	Verificar el rendimiento bajo carga	Simular 500 usuarios registrándose y comprando entradas simultáneamente	El sistema debe manejar la carga sin caídas de rendimiento o errores	Pasó

IV. III. Aplicación de las Pruebas

A continuación, se mostrarán las pruebas realizadas dentro de los sistemas de RegCon, las pruebas contienen del tipo funcionales y no funcionales y se harán estructurados a como se menciona en el esquema ISTQB Certified Tester Foundation Level, pp. 109-130.

Prueba de Registro de Usuario

- ID del Caso de Prueba: TC01

- Objetivo: Verificar que el sistema permite registrar nuevos usuarios.
- Requisitos de Entrada: Nombre, correo electrónico, contraseña.
- Pasos de Prueba:
    1. Acceder a la página de registro.
    2. Completar todos los campos obligatorios.
    3. Hacer clic en "Registrar".
- Resultado Esperado: El usuario debe ser registrado exitosamente y recibir un correo de confirmación.
- Criterios de Aceptación:
    - El sistema crea el usuario y redirige al usuario a la página de inicio de sesión.
    - Se debe recibir un correo de confirmación.
- Estado Actual: Pasó
- Notas: Sin notas adicionales.

Prueba de Compra de Entradas

- ID del Caso de Prueba: TC02
- Objetivo: Verificar que los usuarios puedan comprar entradas para los eventos.
- Requisitos de Entrada: Datos de usuario autenticado, selección de evento y tipo de entrada.
- Pasos de Prueba:
    1. Iniciar sesión con un usuario válido.

2. Navegar hasta la sección de eventos.
3. Seleccionar un evento disponible y hacer clic en "Comprar entrada".
4. Seleccionar el tipo de entrada y completar la compra.

- Resultado Esperado: La compra debe procesarse correctamente, y el ticket debe generarse con la información correspondiente.
- Criterios de Aceptación:
  - La compra debe ser procesada sin errores.
  - El sistema genera un ticket con los datos correctos del evento y categoría de entrada.
- Estado Actual: Pasó.
- Notas: Verificar si el pago es exitoso y que se reflejen los cambios en la aplicación de tercero

Prueba de Autenticación de Usuario (Inicio de Sesión)

- ID del Caso de Prueba: TC03
- Objetivo: Verificar que el sistema permite que los usuarios inicien sesión correctamente.
- Requisitos de Entrada: Correo electrónico y contraseña válidos.
- Pasos de Prueba:
  1. Acceder a la página de inicio de sesión.
  2. Ingresar el correo electrónico y la contraseña.
  3. Hacer clic en "Iniciar sesión".

- Resultado Esperado: El usuario debe acceder al sistema y ser redirigido a la página principal o dashboard.
- Criterios de Aceptación:
    - El usuario debe acceder sin problemas si las credenciales son correctas.
    - El sistema debe mostrar un mensaje de error si las credenciales son incorrectas.
- Estado Actual: Pasó.
- Notas: Verificar si el sistema maneja correctamente los intentos de inicio de sesión fallidos.

Prueba de Validación de Asistencia mediante QR

- ID del Caso de Prueba: TC04
- Objetivo: Verificar que el sistema pueda validar la asistencia de los usuarios mediante un código QR.
- Requisitos de Entrada: Código QR válido de un ticket.
- Pasos de Prueba:
    1. Acceder a la sección de validación de asistencia.
    2. Escanear un código QR de un ticket de usuario.
    3. Verificar si la asistencia se registra correctamente.
- Resultado Esperado: El sistema debe registrar la asistencia y mostrar una confirmación en pantalla.
- Criterios de Aceptación:

    - El sistema debe registrar correctamente la asistencia al evento y actualizar la base de datos.
    - El sistema debe mostrar un mensaje de confirmación de asistencia.

- Estado Actual: Pasó.
- Notas: Asegurarse de que el código QR esté vinculado correctamente a la base de datos.

Prueba de Seguridad: Inyección SQL en Formulario de Registro

- ID del Caso de Prueba: TC05
- Objetivo: Verificar que el sistema esté protegido contra ataques de inyección SQL en el formulario de registro.
- Requisitos de Entrada: Ingreso de código malicioso en el formulario de registro.
- Pasos de Prueba:
    1. Acceder a la página de registro.
    2. Intentar ingresar un comando SQL malicioso en el campo de nombre o correo electrónico (por ejemplo, ' OR 1=1 --).
    3. Hacer clic en "Registrar".
- Resultado Esperado: El sistema debe rechazar el intento de inyección SQL y no permitir el registro.
- Criterios de Aceptación:
    - El sistema debe sanitizar los inputs y prevenir inyecciones SQL.
    - El sistema debe mostrar un mensaje de error adecuado.

- Estado Actual: Pasó.
- Notas: Este tipo de prueba es crucial para garantizar la seguridad de los datos de los usuarios.

Prueba de Rendimiento Bajo Carga (500 Usuarios Simultáneos)

- ID del Caso de Prueba: TC06
- Objetivo: Verificar cómo se comporta el sistema cuando se simulan 500 usuarios accediendo al sistema al mismo tiempo.
- Requisitos de Entrada: Simulación de 500 usuarios que realizan diversas acciones (registro, compra de entradas, validación de asistencia).
- Pasos de Prueba:
    1. Configurar un entorno de pruebas con 500 usuarios simultáneos.
    2. Ejecutar diversos procesos (registro, compra de entradas, validación de asistencia) a la vez.
    3. Medir los tiempos de respuesta y la estabilidad del sistema.
- Resultado Esperado: El sistema debe funcionar sin caídas o errores de servidor.
- Criterios de Aceptación:
    - El sistema debe manejar la carga sin errores de tiempo de respuesta.
    - No debe haber caídas del servidor o bloqueos durante las interacciones.
- Estado Actual: Pasó.
- Notas: Verificar el comportamiento en condiciones de alto tráfico.

Prueba de Usabilidad: Interfaz en Dispositivos Móviles

- ID del Caso de Prueba: TC07
- Objetivo: Verificar que la interfaz del sistema sea funcional y usable en dispositivos móviles.
- Requisitos de Entrada: Dispositivo móvil con acceso a internet.
- Pasos de Prueba:
    1. Acceder al sistema desde un dispositivo móvil.
    2. Navegar por las páginas de registro, eventos y compra de entradas.
    3. Verificar si la interfaz es responsiva y fácil de usar.
- Resultado Esperado: La interfaz debe adaptarse correctamente a las pantallas móviles sin perder funcionalidad.
- Criterios de Aceptación:
    - Todos los elementos deben ser accesibles y la navegación debe ser fluida en pantallas pequeñas.
    - No debe haber problemas visuales o funcionales en dispositivos móviles.
- Estado Actual: Falló.
- Notas: Si bien el sistema funciona bien y se adapta a dispositivos móviles por el Tailwind CSS y React, no están todos los componentes y módulos adaptados u optimizados para ejecutarse en dispositivos móviles.

Prueba de Edición de Evento

- ID del Caso de Prueba: TC08

- Objetivo: Verificar que los administradores puedan editar los detalles de un evento.
- Requisitos de Entrada: Datos del evento y permisos de administrador.
- Pasos de Prueba:
    1. Iniciar sesión como administrador.
    2. Navegar hasta la página de eventos.
    3. Seleccionar un evento existente.
    4. Editar los detalles del evento (nombre, fecha, ubicación, etc.).
    5. Guardar los cambios.
- Resultado Esperado: Los cambios en el evento se guardan correctamente y se actualizan en la base de datos.
- Criterios de Aceptación:
    - El evento debe reflejar los cambios realizados en la interfaz.
    - No debe haber errores al guardar los cambios.
- Estado Actual: Pasó.
- Notas: Asegurarse de que los cambios se apliquen correctamente y sean visibles para todos los usuarios.

Prueba de Eliminación de Evento

- ID del Caso de Prueba: TC09
- Objetivo: Verificar que los administradores puedan eliminar eventos correctamente.
- Requisitos de Entrada: Permisos de administrador, evento existente.

- Pasos de Prueba:
    1. Iniciar sesión como administrador.
    2. Navegar hasta la lista de eventos.
    3. Seleccionar un evento para eliminar.
    4. Confirmar la eliminación del evento.
- Resultado Esperado: El evento seleccionado se elimina correctamente de la base de datos y ya no está disponible para los usuarios.
- Criterios de Aceptación:
    - El evento debe desaparecer de la lista de eventos.
    - No deben quedar datos huérfanos en la base de datos.
- Estado Actual: Pasó.
- Notas: Confirmar que no se puedan eliminar eventos si existen registros de asistencia o ventas asociados.

Prueba de Filtrado de Eventos por Categoría

- ID del Caso de Prueba: TC10
- Objetivo: Verificar que los usuarios puedan filtrar eventos según su categoría.
- Requisitos de Entrada: Categorías de eventos configuradas, eventos asociados.
- Pasos de Prueba:
    1. Acceder a la página de eventos.
    2. Aplicar un filtro de categoría (por ejemplo, "Tecnología", "Música").

3. Verificar que solo se muestren los eventos correspondientes a la categoría seleccionada.

- Resultado Esperado: El sistema debe mostrar solo los eventos que pertenecen a la categoría seleccionada.
- Criterios de Aceptación:
    - Los eventos filtrados deben coincidir con la categoría seleccionada.
    - El filtro debe funcionar correctamente sin mostrar eventos fuera de la categoría elegida.
- Estado Actual: Pasó.
- Notas: Asegurarse de que el filtro sea rápido y eficaz en la presentación de resultados.

Prueba de Protección contra Cross-Site Scripting (XSS)

- ID del Caso de Prueba: TC11
- Objetivo: Verificar que el sistema esté protegido contra ataques de XSS.
- Requisitos de Entrada: Script malicioso insertado en formularios.
- Pasos de Prueba:
    1. Acceder a un formulario de entrada (registro, comentarios, etc.).
    2. Intentar ingresar un script malicioso en los campos (por ejemplo, <script>alert('XSS')</script>).
    3. Enviar el formulario.
- Resultado Esperado: El sistema debe filtrar y sanitizar el script antes de almacenarlo o mostrarlo, evitando la ejecución del código malicioso.

- Criterios de Aceptación:
    - El sistema debe prevenir la ejecución de cualquier script malicioso.
    - Los campos deben ser validados y sanitizados adecuadamente.
- Estado Actual: Pasó.
- Notas: Verificar que los formularios no permitan el ingreso de scripts maliciosos.

Prueba de Autorización de Roles

- ID del Caso de Prueba: TC12
- Objetivo: Verificar que los usuarios con diferentes roles tengan accesos restringidos según su autorización.
- Requisitos de Entrada: Roles configurados (administrador, usuario regular).
- Pasos de Prueba:
    1. Iniciar sesión con un usuario regular.
    2. Intentar acceder a la página de administración de eventos.
    3. Verificar que el acceso esté restringido.
    4. Iniciar sesión como administrador y verificar que se tenga acceso completo.
- Resultado Esperado: Los usuarios regulares no deben tener acceso a funciones administrativas. Solo los administradores deben poder acceder a funciones de gestión de eventos.
- Criterios de Aceptación:
    - Los roles deben estar correctamente implementados, con restricciones adecuadas según el tipo de usuario.

- Los usuarios no autorizados deben recibir un mensaje de error adecuado.

- Estado Actual: Pasó.
- Notas: Comprobar la gestión de sesiones y validación de acceso según el rol del usuario.

Prueba de Tiempo de Respuesta del Sistema (Bajo Carga Normal)

- ID del Caso de Prueba: TC13
- Objetivo: Verificar que el sistema mantenga tiempos de respuesta adecuados bajo una carga normal de usuarios.
- Requisitos de Entrada: Accesos concurrentes de 50 usuarios realizando acciones comunes (registro, búsqueda de eventos).
- Pasos de Prueba:
    1. Simular 50 usuarios simultáneos en el sistema.
    2. Realizar tareas comunes (registro de usuario, navegación, compra de entradas).
    3. Medir el tiempo de respuesta del sistema.
- Resultado Esperado: El sistema debe mantener un tiempo de respuesta adecuado (menos de 3 segundos por acción).
- Criterios de Aceptación:
    - El sistema debe responder en menos de 3 segundos para cada solicitud.
    - No debe haber errores de timeout o sobrecarga del sistema.
- Estado Actual: Pasó.
- Notas: Asegurarse de que la experiencia de usuario no se degrade bajo carga normal.

Prueba de Estrés (Carga Máxima de 1000 Usuarios)

- ID del Caso de Prueba: TC14
- Objetivo: Evaluar la capacidad del sistema para manejar un alto volumen de usuarios simultáneamente sin fallos.
- Requisitos de Entrada: Simulación de 1000 usuarios simultáneos realizando operaciones en el sistema.
- Pasos de Prueba:
    1. Simular la carga de 1000 usuarios concurrentes.
    2. Realizar operaciones como registro, compra de entradas y validación de asistencia simultáneamente.
    3. Monitorear el rendimiento del sistema durante la prueba.
- Resultado Esperado: El sistema debe manejar la carga sin caídas ni fallos críticos.
- Criterios de Aceptación:
    - El sistema debe soportar la carga sin bloquearse o mostrar tiempos de respuesta excesivos.
    - El sistema debe ser capaz de seguir funcionando sin errores graves.
- Estado Actual: Falló.
- Notas: Optimizar la infraestructura del servidor y la base de datos durante la prueba.

Prueba de Navegación en Dispositivos Móviles

- ID del Caso de Prueba: TC15
- Objetivo: Verificar la usabilidad del sistema en dispositivos móviles.
- Requisitos de Entrada: Dispositivo móvil con acceso a internet.
- Pasos de Prueba:
    1. Acceder a la aplicación desde un dispositivo móvil.
    2. Navegar por las secciones de registro, eventos y compra de entradas.
    3. Verificar que todos los elementos de la interfaz sean accesibles y fáciles de usar.
- Resultado Esperado: El sistema debe ser completamente funcional en dispositivos móviles, con una interfaz responsiva.
- Criterios de Aceptación:
    - Todos los botones, enlaces y formularios deben ser fáciles de usar en pantallas pequeñas.
    - La navegación debe ser fluida y sin errores visuales.
- Estado Actual: Falló.
- Notas: Adaptar todos los módulos necesarios para que pueda verse de forma correcta la página si se accede desde un dispositivo móvil.

IV. IV. Seguimiento a los Reportes e Incidencias

El seguimiento a los reportes e incidencias es una actividad esencial dentro del proceso de aseguramiento de calidad en RegCon, ya que permite identificar, registrar, priorizar y resolver problemas detectados durante las pruebas o reportados por usuarios finales. Este proceso

asegura que el sistema se mantenga confiable, funcional y alineado con los requerimientos establecidos, además de garantizar la mejora continua. En este apartado, se describen las estrategias, herramientas, flujos de trabajo y métricas implementadas para gestionar eficazmente los reportes e incidencias en el sistema RegCon

IV. IV. I. Objetivo del Seguimiento de Incidencias

1. Detección y Resolución de Problemas:
    - Garantizar que todos los errores y problemas sean detectados, documentados y resueltos de manera oportuna.
2. Priorización Efectiva:
    - Clasificar las incidencias según su impacto en el sistema y en los usuarios, enfocándose primero en las críticas.
3. Mejora Continua:
    - Analizar las tendencias de los reportes para identificar áreas que requieran optimización o rediseño.
4. Comunicación Transparente:
    - Mantener informados a los interesados (stakeholders) sobre el estado de las incidencias y las acciones tomadas para resolverlas.

IV. IV. II. Herramientas para el Seguimiento de Incidencias

1. Jira:
    - Herramienta principal utilizada para registrar, priorizar y hacer seguimiento a las incidencias.
    - Funcionalidades clave:

        - Creación de tickets con descripciones detalladas.
        - Priorización de incidencias por nivel de severidad.
        - Seguimiento del ciclo de vida de cada incidencia (abierto, en progreso, resuelto, cerrado).

2. GitHub Issues:
    - Utilizado para reportar problemas directamente relacionados con el código fuente.
    - Ventajas:
        - Integración con el repositorio de código.
        - Asignación de problemas a desarrolladores específicos.
3. Sentry:
    - Herramienta de monitoreo de errores en tiempo real.
    - Funcionalidades clave:
        - Captura de excepciones no manejadas en frontend y backend.
        - Registro automático de trazas de errores y contexto del usuario.
4. Slack:
    - Canal de comunicación para notificaciones automáticas sobre errores críticos y progreso en la resolución.

IV. IV. III. Proceso de Seguimiento de Incidencias

El flujo de trabajo para el manejo de reportes e incidencias en RegCon está estructurado en las siguientes etapas:

Plan Final

a. Detección del Problema

- Fuentes principales:
    - Pruebas Internas: Errores detectados por el equipo de QA durante las pruebas funcionales, de integración o de estrés.
    - Monitoreo en Producción: Alertas generadas por herramientas como Sentry para errores en tiempo real.
    - Usuarios Finales: Reportes enviados por usuarios a través de formularios o correo electrónico.

b. Registro del Problema

- Cada incidencia se documenta en Jira o GitHub Issues con la siguiente información:
    - Título Descriptivo: Breve descripción del problema.
    - Descripción Detallada:
        - Pasos para reproducir el problema.
        - Resultados esperados vs. resultados actuales.
        - Entorno en el que ocurrió (dispositivo, navegador, versión del sistema).
    - Prioridad y Severidad:
        - Crítica: Incidencias que afectan funcionalidades principales del sistema (por ejemplo, inicio de sesión).
        - Alta: Problemas que afectan funcionalidades importantes, pero tienen solución temporal.

    - Media: Errores que no bloquean el uso del sistema, pero afectan la experiencia del usuario.
    - Baja: Problemas visuales o de bajo impacto.
  - Evidencia: Capturas de pantalla, videos o logs relevantes.

c. Priorización

- Los tickets se clasifican según su impacto en el sistema y la cantidad de usuarios afectados.
- Ejemplo:
  - Prueba de Usabilidad en Móviles (TC07): Clasificada como prioridad alta debido a su impacto en la experiencia móvil.

d. Asignación

- Cada incidencia se asigna a un desarrollador o equipo específico según su experiencia o área del sistema afectada.

e. Resolución

- Los desarrolladores investigan y corrigen el problema utilizando herramientas de depuración y pruebas adicionales.
- Las soluciones incluyen:
  - Actualización del código fuente.
  - Ajustes en configuraciones del servidor o base de datos.
  - Cambios en la interfaz de usuario.

f. Validación

- El equipo de QA verifica que la solución resuelva el problema sin introducir nuevas fallas.
- Se realizan pruebas adicionales para confirmar que el error ya no ocurre en escenarios similares.

g. Cierre

- Una vez validada la solución, el ticket se marca como resuelto o cerrado.
- Se documenta el cambio en el registro de cambios (CHANGELOG.md) del proyecto.

IV. IV. IV. Métricas de Seguimiento

Para medir la efectividad del manejo de incidencias, se monitorean las siguientes métricas:

1. Tiempo de Resolución:
    - Tiempo promedio desde que se detecta un problema hasta que se resuelve.
    - Meta: Resolver problemas críticos en menos de 24 horas.
2. Cantidad de Incidencias Abiertas:
    - Número total de problemas pendientes en un momento dado.
3. Reapertura de Tickets:
    - Porcentaje de tickets cerrados que se reabren debido a soluciones incompletas.
    - Meta: Mantener este porcentaje por debajo del 5%.
4. Distribución por Severidad:
    - Clasificación de incidencias resueltas según su severidad (crítica, alta, media, baja).

Plan Final

IV. IV. V. Reportes de Incidencias

Título: Problemas con la visualización de la interfaz en dispositivos móviles.

Descripción:

- Al acceder al sistema desde un dispositivo móvil, algunos botones no son visibles en pantallas pequeñas.
- Pasos para reproducir:
    1. Acceder al sistema desde un dispositivo móvil.
    2. Navegar a la página de eventos.
    3. Intentar comprar una entrada.
- Impacto: Los usuarios móviles no pueden completar la compra de entradas.
- Evidencia: Captura de pantalla donde los botones están fuera de la vista.
- Prioridad: Alta.
- Estado Actual: En progreso.
- Notas: Requiere ajuste en estilos CSS para pantallas pequeñas.

IV. IV. VI. Resultados Obtenidos

1. Resolución de Problemas Críticos:
    - Los problemas reportados en pruebas como TC05 (inyección SQL) y TC11 (XSS) se resolvieron rápidamente, asegurando la seguridad del sistema.
2. Mejora Continua:
    - Los problemas de rendimiento bajo carga (TC06 y TC14) llevaron a la optimización del servidor y consultas a la base de datos.

3. Transparencia:
    - Los stakeholders reciben actualizaciones regulares sobre el estado de los problemas gracias al uso de herramientas como Jira.
4. Reducción de Errores Recurrentes:
    - La implementación de registros detallados permitió identificar patrones de errores y evitar su repetición.

V. Liberación de Software

V. I. Liberación de las versiones

La liberación del software es un proceso clave que marca el momento en que el producto desarrollado se pone a disposición de los usuarios, asegurando que cumpla con los estándares de calidad, funcionalidad y estabilidad establecidos. En RegCon, el proceso de liberación de versiones fue cuidadosamente planificado y ejecutado para garantizar un despliegue exitoso tanto del frontend como del backend, además de incluir un seguimiento continuo para garantizar que las actualizaciones futuras se implementen de manera eficiente. A continuación, se detalla cómo se realizó la liberación de las versiones del software, incluyendo las herramientas, flujos de trabajo, estrategias de despliegue y los cambios significativos que se llevaron a cabo durante el desarrollo y mantenimiento del sistema.

V. I. I. Estrategia de Liberación

La liberación de versiones en RegCon siguió una estrategia basada en las siguientes prácticas:

1. Versionado Semántico (SemVer):
    - El sistema utiliza un esquema de versionado semántico (major.minor.patch), donde:

- Major: Cambios que rompen compatibilidad (e.g., reestructuración completa de la base de datos).
- Minor: Nuevas funcionalidades que son compatibles con la versión anterior (e.g., filtrado por categorías).
- Patch: Corrección de errores o mejoras menores (e.g., ajuste de estilos en móviles).

2. Liberaciones Iterativas:
    - Se adoptó un enfoque iterativo, liberando versiones estables en ciclos definidos, permitiendo un desarrollo incremental y validación continua.
3. Pruebas Previas al Despliegue:
    - Cada versión pasa por un riguroso proceso de pruebas (unitarias, integradas y de estrés) antes de ser desplegada.

V. I. II. Herramientas para la Liberación

El despliegue de las versiones del frontend y backend de RegCon se llevó a cabo utilizando las siguientes herramientas y servicios:

1. Backend:
    - Node.js y Express: Base del backend que gestiona las API RESTful.
    - Docker:
        - Utilizado para empaquetar el backend en contenedores, asegurando que el entorno de desarrollo sea idéntico al de producción.
        - Configuración de un Dockerfile optimizado para Back4App.
    - Back4App (Hosting):

        - Plataforma utilizada para desplegar el backend, ofreciendo escalabilidad automática y soporte para PostgreSQL en la nube.

2. Frontend:
    - React: Base del frontend, desarrollado con componentes reutilizables y diseño responsivo.
    - Vercel:
        - Plataforma seleccionada para desplegar el frontend, ofreciendo despliegue automático a partir de cambios en el repositorio de GitHub.
        - Proporciona un dominio personalizado y soporte para pruebas de previsualización.
3. Control de Versiones:
    - GitHub: Repositorio principal para la gestión del código, integrado con pipelines de CI/CD para automatizar pruebas y despliegues.
4. Base de Datos:
    - Supabase: Base de datos relacional alojada en la nube, con soporte para replicación y escalabilidad.
5. Monitoreo y Gestión de Errores:
    - Sentry: Para capturar errores en tiempo real en el backend y frontend, lo que facilita la resolución rápida de problemas después del despliegue.
    - Supabase: Para administrar y monitorear la base de datos PostgreSQL.

V. I. III. Proceso de Liberación

El proceso de liberación de versiones de RegCon siguió una serie de pasos bien definidos:

Plan Final

a. Preparación

1. Revisión de Código:
    - Todo el código fue revisado en GitHub mediante pull requests, asegurando que cumpla con los estándares de calidad.
2. Pruebas Exhaustivas:
    - Las pruebas incluyeron casos de uso críticos, como registro de usuarios (TC01), compra de entradas (TC02) y validación de asistencia (TC04).

b. Despliegue del Backend

1. Empaquetado con Docker:
    - El backend se empaquetó en un contenedor Docker utilizando un Dockerfile optimizado.
    - Configuración:

```
FROM node:16-alpine

WORKDIR /app

COPY package*.json ./

RUN npm install

COPY . .

EXPOSE 3000

CMD ["node", "server.js"]
```

2. Despliegue en Back4App:

- El contenedor se subió y desplegó en Back4App, aprovechando su infraestructura escalable.

3. Pruebas Post-Despliegue:
    - Se verificaron los endpoints críticos del backend utilizando Postman y Supertest.

c. Despliegue del Frontend

1. Integración con GitHub:
    - Vercel se configuró para desplegar automáticamente el frontend cada vez que se fusionaban cambios en la rama main.
2. Dominio Personalizado:
    - Se configuró un dominio personalizado para facilitar el acceso de los usuarios finales.
3. Pruebas de Responsividad:
    - Se verificó que el frontend funcionara correctamente en dispositivos de escritorio y móviles, identificando y ajustando problemas reportados en TC07 (usabilidad móvil).

d. Liberación de la Base de Datos

1. Migraciones:
    - Se realizaron migraciones utilizando herramientas como Sequelize y Knex.js para aplicar cambios en la estructura de la base de datos.
    - Script de migración:

```
exports.up = function(knex) {
```

```
  return knex.schema.createTable('tickets', function(table) {
    table.increments('id').primary();
    table.string('categoria').notNullable();
    table.integer('evento_id').references('id').inTable('eventos');
  });
};
```

2. Pruebas de Integridad:
    - Se ejecutaron consultas para verificar que los datos existentes no se vieran afectados por las migraciones.

e. Comunicación

- Se informó a los usuarios finales y stakeholders sobre la nueva versión mediante correo electrónico y notificaciones en la plataforma.

V. I. IV. Cambios Significativos Durante las Liberaciones

1. Migración de la Base de Datos:
    - Se realizó una migración completa de la base de datos local a una instancia en la nube (PostgreSQL) para mejorar la escalabilidad.
2. Mejoras de Usabilidad Móvil:
    - A partir del reporte de TC07, se realizaron ajustes en los estilos CSS para garantizar que todos los componentes sean responsivos.
3. Optimización de Rendimiento:

- Los problemas identificados en TC06 y TC14 llevaron a optimizar consultas SQL y configurar caché en el backend.

4. Nuevas Funcionalidades:
    - Liberación de filtros por categoría (TC10) y validación mejorada de roles (TC12).

V. I. IV. Resultados Obtenidos

1. Despliegues Automatizados:
    - La integración con Vercel y Back4App permite liberar nuevas versiones en minutos, reduciendo significativamente el tiempo de implementación.
2. Estabilidad:
    - El sistema ha demostrado ser estable bajo cargas normales (TC13), con una disponibilidad superior al 99%.
3. Mejoras Continuas:
    - La recopilación de métricas y el análisis de reportes han permitido implementar mejoras iterativas.
4. Satisfacción del Usuario:
    - La liberación de versiones con funcionalidades completas y sin errores críticos ha incrementado la confianza de los usuarios en la plataforma.

VI. Administración de la Configuración

VI. I. Selección de Herramienta para Control de Versiones

El control de versiones es un componente esencial en el desarrollo y mantenimiento ya que permite gestionar de manera eficiente los cambios realizados en el código fuente, coordinar el trabajo entre desarrolladores, además de mantener un historial completo de todas las

modificaciones. La elección de la herramienta adecuada para el control de versiones fue un paso estratégico para garantizar la estabilidad, trazabilidad y escalabilidad del sistema.

A continuación, se describe el proceso de selección, configuración y uso de la herramienta de control de versiones adoptada en RegCon, detallando sus beneficios, integración con el flujo de trabajo del proyecto y su impacto en la administración de la configuración.

VI. I. I. Herramienta Seleccionada: Git y GitHub

Git fue seleccionado como el sistema de control de versiones para RegCon, mientras que GitHub se eligió como la plataforma de alojamiento remoto. Esta combinación se basó en su flexibilidad, popularidad en la industria, y el extenso soporte de herramientas complementarias.

VI. I. II. Razones para la Selección de Git

Git se seleccionó debido a sus características avanzadas y su capacidad para manejar proyectos de cualquier tamaño de manera eficiente. Algunos de los factores clave que influyeron en su elección incluyen:

1. Sistema Distribuido:
    - Cada desarrollador tiene una copia completa del repositorio, lo que permite trabajar de forma independiente sin necesidad de una conexión constante a Internet.
    - Esto facilita la colaboración en equipo y asegura que los datos estén protegidos contra fallos en el servidor central.
2. Gestión de Ramas:
    - Git proporciona un sistema de ramas liviano y eficiente, lo que permite desarrollar nuevas funcionalidades, corregir errores y realizar pruebas en entornos aislados antes de integrarlas al código principal.

- Ejemplo de ramas en RegCon:
  - main: Contiene la versión estable en producción.
  - develop: Rama base para el desarrollo de nuevas funcionalidades.
  - feature/<nombre>: Ramas para funcionalidades específicas (e.g., feature/validacion-qr).
  - hotfix/<nombre>: Ramas para correcciones urgentes en producción.

3. Rendimiento:
   - Git maneja eficientemente proyectos grandes, ofreciendo un rendimiento rápido en operaciones como commits, diffs y fusiones.
4. Historial Detallado:
   - Git almacena un historial completo de todos los cambios realizados, incluyendo autor, fecha y descripción, lo que facilita la trazabilidad.
5. Amplio Ecosistema:
   - Git se integra con herramientas como Jira, VSCode, Docker, y servicios de CI/CD como GitHub Actions.

VI. I. III. Razones para la Selección de GitHub

GitHub fue elegido como la plataforma de alojamiento debido a las siguientes características:

1. Almacenamiento Centralizado:
   - Permite almacenar el código fuente en un lugar seguro, accesible para todo el equipo.
2. Colaboración y Revisión de Código:

- GitHub facilita la colaboración mediante pull requests, revisiones de código y comentarios en los cambios realizados.

3. Integración con CI/CD:
    - GitHub Actions se utilizó para automatizar pruebas y despliegues.
    - Ejemplo de flujo automatizado:
        - Cada commit en main dispara un pipeline que:
            - Ejecuta pruebas unitarias e integradas.
            - Construye la aplicación.
            - Despliega el frontend en Vercel y el backend en Back4App.
4. Gestión de Issues:
    - Se utiliza para registrar y rastrear errores e incidencias reportadas por el equipo o los usuarios finales.
5. Seguridad:
    - GitHub asegura el repositorio mediante controles de acceso, autenticación en dos pasos y protección de ramas.

VI. I. IV. Configuración Inicial

La configuración del control de versiones para RegCon se realizó siguiendo estos pasos:

1. Inicialización del Repositorio:
    - El repositorio fue inicializado con Git en local y luego conectado a GitHub:

```
git init

git remote add origin https://github.com/Darkyapper/regcon.git
```

2. Estructura de Ramas:
    - Se definió la estructura de ramas mencionada anteriormente para garantizar un flujo de trabajo organizado.
3. Hooks de Git:
    - Se implementaron hooks locales, como pre-commit, para validar el código antes de cada commit:

```
# pre-commit hook
npm run lint
```

4. Protección de Ramas:
    - En GitHub, la rama main fue protegida, requiriendo revisiones obligatorias antes de permitir fusiones.
5. Automatización con GitHub Actions:
    - Se configuró un archivo YAML para automatizar pruebas y despliegues:

```
name: CI/CD Pipeline
on:
  push:
    branches:
      - main
jobs:
  build-and-test:
```

```
    runs-on: ubuntu-latest

    steps:

      - name: Checkout code

        uses: actions/checkout@v2

      - name: Set up Node.js

        uses: actions/setup-node@v2

        with:

          node-version: '16'

      - name: Install dependencies

        run: npm install

      - name: Run tests

        run: npm test
```

VI. I. V. Flujos de Trabajo Implementados

El equipo adoptó los siguientes flujos de trabajo para el control de versiones:

1. Git Flow Modificado:
    - Los desarrolladores trabajan en ramas independientes y envían pull requests para integrar los cambios.
    - Ejemplo de flujo:
        1. Crear una nueva rama para una funcionalidad:

```
git checkout -b feature/validacion-qr
```

2. Realizar cambios y confirmar:

git add .

git commit -m "feature: implementar validación de QR"

3. Enviar los cambios al repositorio remoto:

git push origin feature/validacion-qr

4. Crear una pull request en GitHub para revisión y fusión.

2. Revisión y Pruebas Automatizadas:
    - Antes de fusionar cambios, se ejecutan pruebas automáticas para garantizar la estabilidad del sistema.
3. Despliegues Automatizados:
    - Las actualizaciones en la rama main desencadenan un despliegue automático en Vercel y Back4App.

VI. I. VI. Beneficios del Control de Versiones con Git y GitHub

1. Colaboración Eficiente:
    - Los desarrolladores pueden trabajar simultáneamente sin conflictos, gracias a la gestión de ramas.
2. Trazabilidad:
    - Cada cambio en el código tiene un historial completo, incluyendo quién lo realizó y por qué.
3. Automatización:

- Los procesos de pruebas y despliegues están completamente automatizados, reduciendo errores humanos.

4. Seguridad:
    - Los controles de acceso y la protección de ramas aseguran que solo el código revisado y aprobado llegue a producción.
5. Escalabilidad:
    - La infraestructura basada en Git y GitHub soporta el crecimiento del proyecto y el equipo.