
# Alquifiestas: Sensacional Eventos
# Sprint 9

Hugo Eduardo Rivas Fajardo - 22500
José Santiago Pereira Alvarado - 22318
Nancy Gabriela Mazariegos Molina - 22513
Giovanni Alejandro Santos Hernández - 22523
Mauricio Julio Rodrigo Lemus Guzmán- 22461
Alexis Mesías Flores - 22562

**Product Backlog**

**Pila del producto**
https://docs.google.com/spreadsheets/d/1pHFa_t3TDiU_gIoxtiY6VaneJB60KE6gEEqGTWLpKHg/edit?gid=1739314590#gid=1739314590

**Gestión de tareas**

https://docs.google.com/spreadsheets/d/1pHFa_t3TDiU_gIoxtiY6VaneJB60KE6gEEqGTWLpKHg/edit?gid=0#gid=0

**Sprint Backlog**

https://docs.google.com/spreadsheets/d/1pHFa_t3TDiU_gIoxtiY6VaneJB60KE6gEEqGTWLpKHg/edit?gid=1378010058#gid=1378010058

**Dashboard:**

https://www.canva.com/design/DAGTmvTELKw/MjHZXHdYUqNPHTn_qwFX7A/edit?utm_content=DAGTmvTELKw&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton

**Incremento**

- Código desarrollado
    - Vínculo al repositorio: https://github.com/Rodlemus03/SensacionalEventos
- Lista de funcionalidades planificadas que se terminaron completamente.
    - Creación de página de notificaciones de pedidos pendientes.
    - Creación de tablas en la BD de las notificaciones.
    - Adecuación de servicios en backend.
    - Creación de disparador .bat

**Prueba**

**Master plan de unit test:**

1. Configuración Inicial

Se usa selenium para la automatización de las pruebas funcionales del sistema. Se crea el modelo propuesto en PostgreSql para almacenar los resultados de las pruebas, ya sean exitosas, fallidas, ignoradas, etc. Al finalizar cada prueba se guarda el resultado esperado vs el resultado real, para poder cotejar la calidad del software.

FrameWorks:

- Selenium
- PsycoPg2

Herramientas externas

- Postman
- PgAdmin
- GitHub actions

2. Plan de Pruebas

**Clientes (CRUD)**

Create Cliente:

Prueba de éxito:

- Enviar una solicitud POST con datos válidos para crear un cliente.
- Verificar que el cliente se creó correctamente en la base de datos.
- Guardar el resultado de la prueba en la base de datos.

Prueba de fallo:

- Enviar una solicitud POST con datos inválidos (e.g., campos obligatorios faltantes).
- Verificar que se recibe una respuesta de error adecuada.
- Guardar el resultado de la prueba en la base de datos.

Read Cliente:

Prueba de éxito:

- Enviar una solicitud GET para recuperar un cliente existente.
- Verificar que los datos del cliente sean correctos.
- Guardar el resultado de la prueba en la base de datos.

Prueba de fallo:

- Enviar una solicitud GET para recuperar un cliente inexistente.
- Verificar que se recibe una respuesta de error adecuada.
- Guardar el resultado de la prueba en la base de datos.

Update Cliente:

Prueba de éxito:

- Enviar una solicitud PUT con datos válidos para actualizar un cliente.
- Verificar que los datos del cliente se actualizaron correctamente en la base de datos.
- Guardar el resultado de la prueba en la base de datos.

Prueba de fallo:

- Enviar una solicitud PUT con datos inválidos.
- Verificar que se recibe una respuesta de error adecuada.
- Guardar el resultado de la prueba en la base de datos.

Delete Cliente:

Prueba de éxito:

- Enviar una solicitud DELETE para eliminar un cliente existente.
- Verificar que el cliente se eliminó correctamente de la base de datos.
- Guardar el resultado de la prueba en la base de datos.

Prueba de fallo:

- Enviar una solicitud DELETE para eliminar un cliente inexistente.
- Verificar que se recibe una respuesta de error adecuada.
- Guardar el resultado de la prueba en la base de datos.

**Inventario (CRUD)**

Create Inventario:

Prueba de éxito:

- Enviar una solicitud POST con datos válidos para crear un ítem de inventario.
- Verificar que el ítem se creó correctamente en la base de datos.
- Guardar el resultado de la prueba en la base de datos.

Prueba de fallo:

- Enviar una solicitud POST con datos inválidos.
- Verificar que se recibe una respuesta de error adecuada.
- Guardar el resultado de la prueba en la base de datos.

Read Inventario:

Prueba de éxito:

- Enviar una solicitud GET para recuperar un ítem de inventario existente.
- Verificar que los datos del ítem sean correctos.
- Guardar el resultado de la prueba en la base de datos.

Prueba de fallo:

- Enviar una solicitud GET para recuperar un ítem inexistente.
- Verificar que se recibe una respuesta de error adecuada.
- Guardar el resultado de la prueba en la base de datos.

Update Inventario:

Prueba de éxito:

- Enviar una solicitud PUT con datos válidos para actualizar un ítem de inventario.
- Verificar que los datos del ítem se actualizaron correctamente en la base de datos.
- Guardar el resultado de la prueba en la base de datos.

Prueba de fallo:

- Enviar una solicitud PUT con datos inválidos.
- Verificar que se recibe una respuesta de error adecuada.
- Guardar el resultado de la prueba en la base de datos.

Delete Inventario:

Prueba de éxito:

- Enviar una solicitud DELETE para eliminar un ítem de inventario existente.
- Verificar que el ítem se eliminó correctamente de la base de datos.
- Guardar el resultado de la prueba en la base de datos.

Prueba de fallo:

- Enviar una solicitud DELETE para eliminar un ítem inexistente.
- Verificar que se recibe una respuesta de error adecuada.
- Guardar el resultado de la prueba en la base de datos.

**Pedidos (CRUD)**

Create Pedido:

Prueba de éxito:

- Enviar una solicitud POST con datos válidos para crear un pedido.
- Verificar que el pedido se creó correctamente en la base de datos.
- Guardar el resultado de la prueba en la base de datos.

Prueba de fallo:

- Enviar una solicitud POST con datos inválidos.
- Verificar que se recibe una respuesta de error adecuada.
- Guardar el resultado de la prueba en la base de datos.

Read Pedido:

Prueba de éxito:

- Enviar una solicitud GET para recuperar un pedido existente.
- Verificar que los datos del pedido sean correctos.
- Guardar el resultado de la prueba en la base de datos.

Prueba de fallo:

- Enviar una solicitud GET para recuperar un pedido inexistente.
- Verificar que se recibe una respuesta de error adecuada.
- Guardar el resultado de la prueba en la base de datos.

Update Pedido:

Prueba de éxito:

- Enviar una solicitud PUT con datos válidos para actualizar un pedido.
- Verificar que los datos del pedido se actualizaron correctamente en la base de datos.
- Guardar el resultado de la prueba en la base de datos.

Prueba de fallo:

- Enviar una solicitud PUT con datos inválidos.
- Verificar que se recibe una respuesta de error adecuada.
- Guardar el resultado de la prueba en la base de datos.

Delete Pedido:

Prueba de éxito:

- Enviar una solicitud DELETE para eliminar un pedido existente.

- Verificar que el pedido se eliminó correctamente de la base de datos.
- Guardar el resultado de la prueba en la base de datos.

Prueba de fallo:

- Enviar una solicitud DELETE para eliminar un pedido inexistente.
- Verificar que se recibe una respuesta de error adecuada.
- Guardar el resultado de la prueba en la base de datos.

3. Automatización con Selenium

Login:

- Verificar que el usuario pueda iniciar sesión correctamente.
- Verificar que los intentos de inicio de sesión fallidos muestren mensajes de error adecuados.
- Guardar el resultado de cada prueba en la base de datos.

Navegación:

- Verificar que los usuarios puedan navegar entre las diferentes páginas (Clientes, Inventario, Pedidos) correctamente.
- Guardar el resultado de cada prueba en la base de datos.

Formularios:

- Verificar que los formularios de creación, actualización y eliminación funcionen correctamente para cada módulo (Clientes, Inventario, Pedidos).
- Guardar el resultado de cada prueba en la base de datos.

4. Ejecución de Pruebas

- Crear scripts de Postman para todas las pruebas de API.
- Utilizar Pytest para organizar y ejecutar las pruebas.
- Configurar Selenium para automatizar las pruebas de interfaz de usuario.
- Registrar los resultados de cada prueba en la base de datos.
- Ejecutar las pruebas en un entorno de CI/CD para garantizar que las pruebas se ejecuten automáticamente con cada cambio en el código.

5. Informe de Resultados

- Generar informes detallados con los resultados de las pruebas.
- Incluir información sobre las pruebas que fallaron, junto con los detalles del error.
- Incluir referencias a los registros almacenados en la base de datos.
- Revisar y corregir los errores identificados durante las pruebas.

**Pruebas de seguridad**

Los resultados obtenidos en general fueron los esperados, ya que llevamos a cabo dos pruebas de seguridad clave para evaluar la seguridad de nuestra aplicación. Estas pruebas estuvieron enfocadas en verificar la capacidad de la aplicación para resistir ataques comunes como inyecciones SQL y accesos no autorizados. En la primera prueba, simulamos inyecciones SQL mediante la inserción de comandos en los formularios de entrada, validando que el sistema sea capaz de filtrar y gestionar adecuadamente los datos sin comprometer su integridad o permitir la alteración no deseada de la base de datos. La segunda prueba se centró en evitar accesos no permitidos, asegurando que los usuarios sin las credenciales adecuadas no pudieran acceder a áreas restringidas o realizar acciones no autorizadas. Ambas pruebas confirmaron la fiabilidad y seguridad de nuestra aplicación, demostrando que las medidas de protección implementadas son efectivas para salvaguardar tanto los datos como el correcto funcionamiento del sistema.

**Pruebas de volumen**

Como parte de las pruebas de volumen, se llevaron a cabo evaluaciones exhaustivas para medir la eficiencia del sistema bajo condiciones de carga significativa. Inicialmente, se realizó una simulación que involucró cincuenta usuarios concurrentes, cada uno interactuando con las distintas funcionalidades del sistema y consumiendo diversos endpoints del API. Durante esta prueba, cada usuario generó un total de 5000 requests, lo que permitió observar el comportamiento del sistema al procesar grandes volúmenes de peticiones en un corto periodo de tiempo.

Los resultados fueron altamente satisfactorios. A pesar de la carga considerable, el sistema mantuvo un tiempo de respuesta constante y eficiente, procesando cada solicitud en tan solo cuatro milisegundos. Este rendimiento demostró la capacidad del sistema para gestionar de manera efectiva un alto volumen de usuarios y solicitudes simultáneas sin experimentar una degradación significativa en su desempeño. Además, no se observaron cuellos de botella ni problemas relacionados con la saturación de recursos, lo que confirma que la infraestructura y la arquitectura del sistema están optimizadas para soportar escenarios de uso intensivo.

Este tipo de pruebas es crucial para garantizar que el sistema pueda escalar adecuadamente en situaciones de producción real, donde se espera que múltiples usuarios interactúen con la aplicación de forma simultánea. La estabilidad y rapidez de respuesta bajo cargas elevadas validan la robustez y capacidad del sistema para ofrecer un servicio confiable incluso en situaciones de alto tráfico.

**Refactorización**

Durante este y los sprints restantes, se ha planificado la refactorización del código con el objetivo de reducir la deuda técnica acumulada y mitigar los riesgos de seguridad presentes. En este sprint en particular, se enfocó la refactorización en varias áreas clave del código que presentaban problemas de mantenibilidad y potenciales vulnerabilidades.

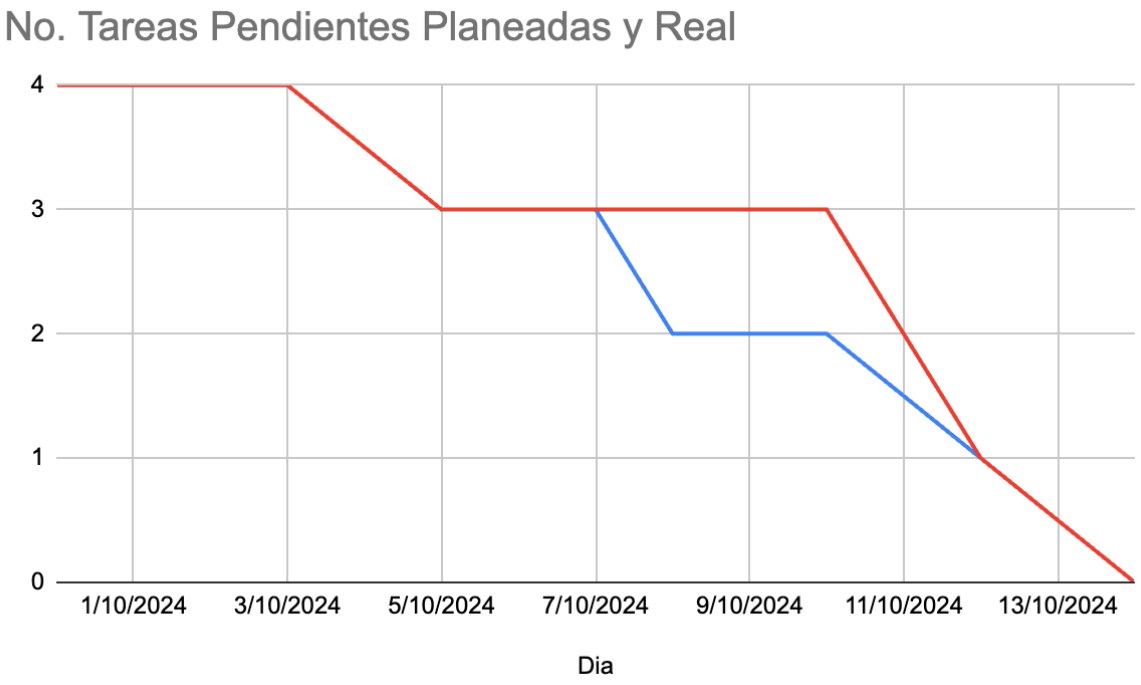
Primero, se mejoró la estructura de los módulos encargados de la autenticación y autorización de usuarios, que originalmente contenían código duplicado y difícil de escalar. Al refactorizar esta parte del código, no solo se eliminó la redundancia, sino que también se fortalecieron los controles de acceso, minimizando el riesgo de brechas de seguridad. Esta refactorización incluyó la implementación de patrones de diseño más robustos, como el uso de middleware para verificar las credenciales de acceso en cada endpoint, lo que facilita futuras expansiones y reduce el riesgo de errores en nuevas funcionalidades.

Además, se revisaron las consultas a la base de datos para garantizar que estuvieran protegidas contra posibles inyecciones SQL. El código que manejaba estas consultas fue optimizado, reemplazando las consultas construidas de forma dinámica por consultas preparadas, lo cual mejora tanto el rendimiento como la seguridad del sistema.

Esta refactorización ha contribuido significativamente a reducir la deuda técnica, al hacer el código más legible, modular y fácil de mantener, lo que a su vez facilita futuras actualizaciones y reduce la probabilidad de errores en el desarrollo a largo plazo. También se ha fortalecido la seguridad del sistema al abordar riesgos previamente identificados, asegurando que las mejoras no solo sean visibles a nivel de performance, sino también en la protección de los datos y la integridad del sistema.

**Resultados**

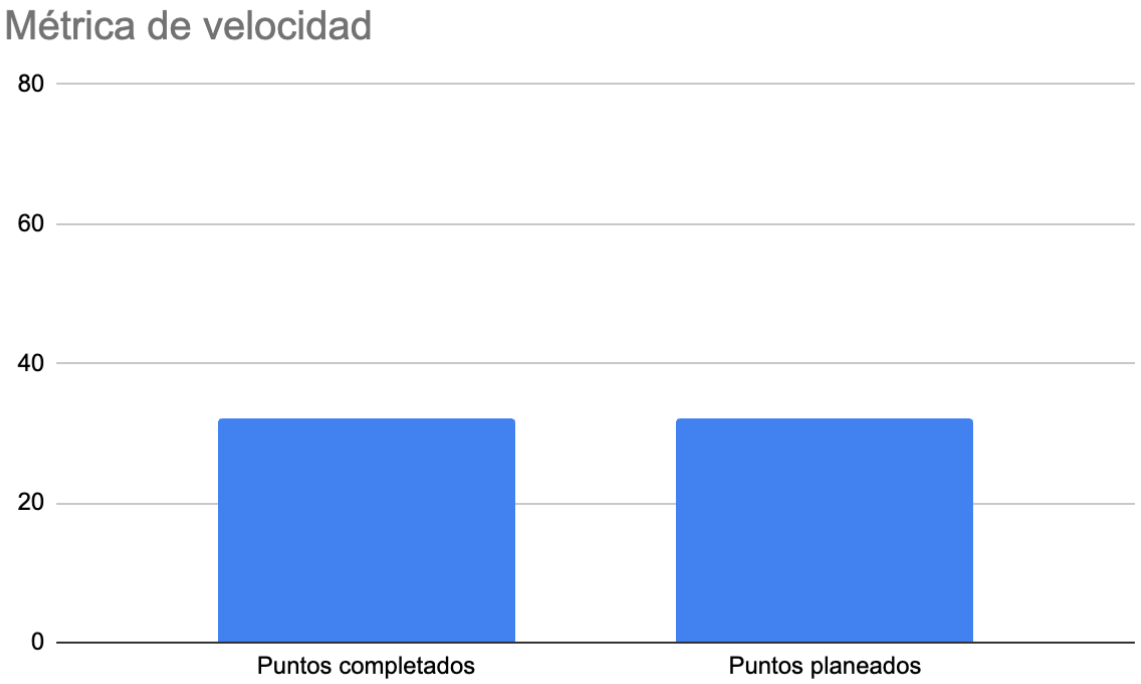
- **Gráfico burndown**



- **Métrica de velocidad**

  **Puntos de historia planificados: 32**

  **Puntos de historia terminados: 32**




**Velocidad del equipo en el sprint** $= \frac{32}{32} \; - \; 0 \; = \; 100\%$

- Como se muestra en los gráficos presentados anteriormente, el equipo logró cumplir con los objetivos planteados para este sprint. Se tenían asignadas cuatro tareas, y todas fueron completadas dentro del tiempo previsto.


- **Presupuesto**

| Descripción | Monto |
|---|---|
| Server EC2 | $ 20.00 |
| Dominio | $ 1.40 |
| Salarios (6 empleados) | $ 2,240.00 |
| Sistema Declaraguate | $ 20.00 |
| Renta de 6 Computadoras | $ 1,200.00 |
| Backup Físico | $ 50.00 |
| Internet | $ 50.00 |
| Total | $ 3,581.40 |

- **Retrospectiva del sprint.**

Durante este sprint, logramos cumplir con todos los objetivos establecidos, alcanzando resultados muy satisfactorios en las áreas de pruebas de seguridad, pruebas de volumen y refactorización del código. En cuanto a las pruebas de seguridad, se realizaron dos evaluaciones clave que confirmaron la capacidad de la aplicación para resistir ataques como inyecciones SQL y accesos no autorizados. Ambas pruebas demostraron la fiabilidad de nuestra solución al garantizar que los datos permanecen protegidos y el sistema funciona de manera segura. En las pruebas de volumen, se simularon cincuenta usuarios concurrentes, generando un total de 5000 solicitudes por usuario, y el sistema respondió de manera eficiente, procesando cada solicitud en apenas cuatro milisegundos. Esto validó la capacidad del sistema para manejar altos volúmenes de tráfico sin sufrir degradaciones en su rendimiento.

En términos de refactorización, se trabajó para reducir la deuda técnica y fortalecer la seguridad. La reorganización de los módulos de autenticación y autorización, junto con la optimización de las consultas a la base de datos mediante el uso de consultas preparadas, mejoró significativamente la mantenibilidad y seguridad del código. Estas mejoras no solo hicieron que el sistema sea más fácil de escalar y mantener, sino que también redujeron los riesgos de errores. A nivel general, la velocidad se mantuvo constante durante sprint, lo que permitió cumplir con todas las tareas planificadas, incluyendo la inspectoría de seguridad, el monitoreo de estrés, la corrección de accesibilidad y la implementación de la lista de notificaciones. La organización y la gestión efectiva del tiempo fueron factores clave para alcanzar estos resultados y mantener el ritmo de trabajo en todo momento.

**Conclusiones:**

- La velocidad para este Sprint fue bastante buena, esta no disminuyó, se mantuvo constante lo cual fue bueno ya que no se ha perdido el ritmo de trabajo.
- Se logró cumplir lo que se había planeado para este Sprint: inspectoría de seguridad, monitoreo de estrés, corrección de accesibilidad y la lista de notificaciones
- La organización ha sido clave para lograr todos los puntos de este Sprint, por otro lado es esencial tomar en cuenta la gestión de tiempo para estos puntos.

**Gestión del tiempo**
**Nombre:** Mauricio Julio Rodrigo Lemus Guzmán
**Carné:** 22461

| Fecha | Inicio | Fin | Tiempo Interrupción | Delta tiempo | Fase | Comentarios |
|---|---|---|---|---|---|---|

| | | | | | | |
|---|---|---|---|---|---|---|
| 2/10 | 9:00 | 10:10 | 10 mins | 1 hora | Sprint 9 | Inspectoría de seguridad |
| 7/10 | 9:30 | 10:30 | 20 mins | 40 mins | Sprint 9 | Inspectoría de seguridad |
| 12/10 | 8:15 | 9:00 | 10 mins | 35 mins | Sprint 9 | Inspectoría de seguridad |
| 14/10 | 12:00 | 12:40 | 15 mins | 25 mins | Sprint 9 | Inspectoría de seguridad |

**Nombre:** José Santiago Pereira Alvarado
**Carné:** 22318

| **Fecha** | **Inicio** | **Fin** | **Tiempo Interrupción** | **Delta tiempo** | **Fase** | **Comentarios** |
|---|---|---|---|---|---|---|
| 10/10 | 19:00 | 20:00 | 10 min | 50 min | Sprint 9 | Monitoreo de estrés |
| 11/10 | 21:00 | 22:30 | 5 min | 1:25 min | Sprint 9 | Monitoreo de estrés |
| 12/10 | 22:00 | 23:00 | 0 min | 1:00 hr | Sprint 9 | Monitoreo de estrés |

**Nombre:** Nancy Gabriela Mazariegos Molina
**Carné:** 22513

| **Fecha** | **Inicio** | **Fin** | **Tiempo Interrupción** | **Delta tiempo** | **Fase** | **Comentarios** |
|---|---|---|---|---|---|---|
| 10/10 | 16:00 | 16:45 | 25 min | 20 min | Sprint 9 | Notificaciones |
| 11/10 | 12:00 | 13:10 | 10 min | 1 hora | Sprint 9 | Documento |
| 12/10 | 18:00 | 19:15 | 15 min | 1 hora | Sprint 9 | Dashboard |

**Nombre:** Hugo Eduardo Rivas Fajardo
**Carné:** 22500

| **Fecha** | **Inicio** | **Fin** | **Tiempo Interrupción** | **Delta tiempo** | **Fase** | **Comentarios** |
|---|---|---|---|---|---|---|
| 4/10 | 10:00 | 11:00 | 20 mins | 40 mins | Sprint 9 | Inspectoría de seguridad |
| 6/10 | 18:00 | 18:30 | 5 mins | 25 mins | Sprint 9 | Inspectoría de seguridad |

| | | | | | | |
|---|---|---|---|---|---|---|
| 10/10 | 15:00 | 15:40 | 10 mins | 30 mins | Sprint 9 | Inspectoría de seguridad |
| 13/10 | 20:00 | 21:00 | 20 mins | 40 mins | Sprint 9 | Inspectoría de seguridad |

**Nombre:** Giovanni Alejandro Santos Hernández
**Carné:** 22523

| **Fecha** | **Inicio** | **Fin** | **Tiempo Interrupción** | **Delta tiempo** | **Fase** | **Comentarios** |
|---|---|---|---|---|---|---|
| 8/10/24 | 10:30 | 11:30 | 10 min | 50 min | Sprint 9 | Corrección de accesibilidad |
| 9/10/24 | 14:00 | 14:35 | 5 min | 30 min | Sprint 9 | Corrección de accesibilidad |
| 10/10/2024 | 12:30 | 1:05 | 5 min | 30 min | Sprint 9 | Corrección de accesibilidad |

**Nombre:** Alexis Mesías Flores
**Carné:** 22562

| **Fecha** | **Inicio** | **Fin** | **Tiempo Interrupción** | **Delta tiempo** | **Fase** | **Comentarios** |
|---|---|---|---|---|---|---|
| 6/10 | 17:00 | 17:30 | 5 min | 25 min | Sprint 9 | Vista de notificaciones |
| 10/10 | 15:00 | 15:40 | 10 min | 30 min | Sprint 9 | Vista de notificaciones |
| 13/10 | 20:00 | 21:00 | 10 min | 50 min | Sprint 9 | Vista de notificaciones |

## Referencias


- *Postman Collections: Organize API Development and Testing*. (s. f.). Postman API Platform. https://www.postman.com/collection/

- García, B. (s. f.). *Boni García*. https://bonigarcia.dev/