Zundada

Laura Alejandra Páez Daza, Marlon David Pabón Muñoz, María Paula Román Arévalo

No. de Equipo Trabajo: 9

I. INTRODUCCIÓN

En este documento se plantea una solución integral al desafío de la organización de eventos en los ámbitos universitarios, a través del proyecto "Zundada". En su contenido, se explorará a fondo el problema que enfrenta la comunidad universitaria al planificar y gestionar eventos, destacando la falta de una plataforma centralizada, problemas de comunicación, gestión de asistentes y otras complicaciones. Se identificarán los usuarios clave del software y se detallarán sus necesidades, seguido por una descripción exhaustiva de los requisitos funcionales del software. Se ofrecerá una vista preliminar de la interfaz de usuario y se describirán los entornos de desarrollo y operación. Además, se presentará un prototipo inicial del software y se explicará cómo se aplican estructuras de datos para mejorar el rendimiento. Finalmente, se llevarán a cabo pruebas y análisis comparativos para evaluar la eficacia del prototipo. Este documento proporcionará una visión completa de cómo "Zundada" abordará y resolverá los desafíos en la organización de eventos, con el objetivo de mejorar la experiencia de las comunidades universitarias.

II. DESCRIPCIÓN DEL PROBLEMA A RESOLVER

El ámbito universitario es conocido por ser un espacio lleno de actividades y eventos que van desde conferencias académicas hasta actividades sociales y culturales. Sin embargo, la organización de estos eventos puede resultar desafiante y caótica para las personas que organizan estos. Problemas como la falta de una plataforma centralizada para la planificación y gestión de eventos, dificultades en la comunicación con los asistentes, limitaciones en la difusión de eventos y la falta de herramientas eficientes para el seguimiento y control de las actividades planificadas son recurrentes.

Este proceso de organización de eventos dentro de la universidad puede llevar a situaciones como eventos sobreexplotados o subutilizados, problemas con la confirmación de asistencia, falta de información actualizada sobre eventos futuros, entre otros. Este desafío afecta tanto a los organizadores de eventos como a los asistentes, dificultando la participación y la creación de una comunidad universitaria más cohesionada.

El proyecto "Zundada" tiene como objetivo general resolver el problema de la organización de eventos en la Universidad Nacional a través de una plataforma web. El propósito principal es crear un ecosistema en línea que simplifique y mejore la planificación, promoción y gestión de eventos en el ámbito universitario, comenzando con un enfoque en la organización de fiestas y actividades sociales.

Se basará en el uso eficiente de estructuras de datos y algoritmos para gestionar la información relacionada con los eventos y proporcionar una experiencia óptima para los usuarios. Al lograr estos objetivos, se espera crear un entorno más colaborativo y participativo en la Universidad Nacional, donde los eventos se organicen y disfruten de manera más efectiva, contribuyendo así a fortalecer la vida universitaria y fomentar los ámbitos sociales y culturales.

III. USUARIOS DEL PRODUCTO DE SOFTWARE

Administradores
Es un perfil dedicado a los individuos que mantienen y se aseguran de que la plataforma está funcionando correctamente. Sus funciones consisten en revisar el código, adaptarlo de acuerdo a las necesidades, y actualizar la plataforma para garantizar su eficiencia y rapidez.

Debido a que son los encargados de controlar cada detalle de la plataforma, cuentan con los privilegios de acceso y seguridad a cualquier información relacionada con la página y su estructura.

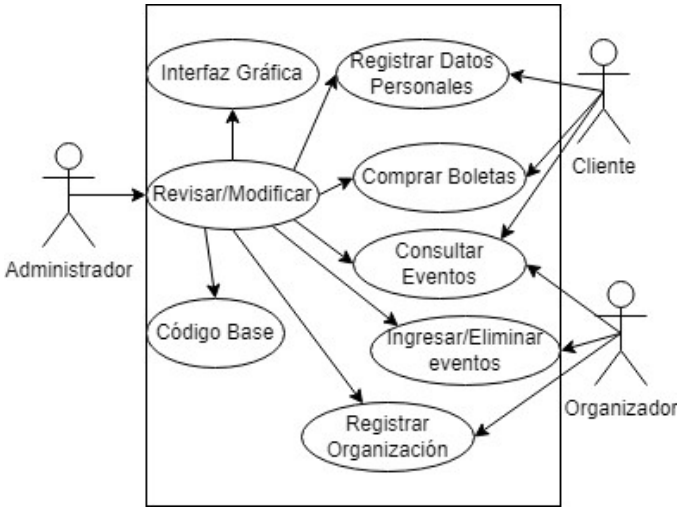
Clientes
Son las personas que utilizan el aplicativo para revisar los eventos disponibles, sus características, y de acuerdo al interés que presenten compran boletas para asistir al evento. Solo acceden a la parte gráfica porque se limitan a acciones como iniciar sesión, actualizar su información personal, comprar boletas, y revisar los eventos (disponibles, agotados o pasados)

Este tipo de rol, no requiere ninguna experiencia, y su frecuencia de uso del aplicativo depende de sus intereses personales.

Organizador
Es una denominación al organizador del evento, se identifica por ser una persona que representa a la entidad que quiere vender boletas y dar a conocer el evento. Este organizador tiene derecho a acceder a la plataforma, con los datos de su organización, y puede colocar los eventos que desee indicando un precio, lugar, fecha, hora, número de boletas disponibles y un dato de contacto. El dato de contacto ya sea número/correo es necesario en caso de que el usuario tenga una pregunta del evento pueda contactarse directamente con el organizador.
Además, cuando un cliente haga su compra, el organizador podrá ver su información básica (nombre, cédula y correo/teléfono de contacto) para llevar un registro, confirmar

la compra de su boleta y contactarse con el usuario en caso de un inconveniente con el evento.

Diagrama de Casos
A continuación se muestra un diagrama que describe en gran medida las funciones de cada usuario, y cómo se relacionan con los demás. Además de que permite ver la información a la que pueden acceder según sus privilegios.



IV. REQUERIMIENTOS FUNCIONALES DEL SOFTWARE

Para el módulo dedicado a la gestión de eventos se deben poder realizar las siguientes funciones:

- **Crear evento:**

La funcionalidad deberá crear un nuevo evento y almacenarlo en la estructura de datos más adecuada.
Únicamente los administradores podrán acceder a esta funcionalidad después de rellenar un formulario con los datos de la fecha, ubicación y número de boletas disponibles del evento.
Requerimientos funcionales:

- Verificar que toda la información cumpla con el tipo de dato aceptado y todos los campos estén completos, en caso contrario mandará una advertencia.
- Comprobar que el evento no se haya registrado anteriormente.

- **Comprar boleta:**

Para todos aquellos usuarios que quieran asistir a un evento, podrán apartar su boleta por medio de esta funcionalidad, donde por medio de una nueva pestaña confirmara el número de boletas y lo redirecciona hacia el chat directo con el organizador de ese evento.
Requerimientos funcionales:

- Permitir seleccionar el número de boletas que se quieran comprar.
- La funcionalidad redirige al usuario a un chat con el o los organizadores solicitando la compra de las boletas.

- **Consulta de eventos registrados:**

En una ventana nueva el programa deberá permitir consultar los eventos vigentes registrados.
Todos los roles de usuarios pueden acceder a esta función.

Requerimientos funcionales:

- Ordenamiento de eventos por el orden de llegada, el número de boletas disponibles o por la proximidad de la fecha.

- **Eliminar evento:**

Saca de la estructura de datos de almacenamiento un evento en específico.
Los administradores seleccionan el evento a eliminar de una lista tipo consulta.
Requerimientos funcionales:

- Si el evento seguía vigente, se envía un mensaje con la cancelación del evento a las personas que tenían comprada su boleta y la devolución de dinero.

- **Modificación datos asociados a los eventos:**

Permite editar alguno de los datos del evento.
Solo los administradores pueden acceder al formulario de modificación.

Requerimientos funcionales:

- Verificar que los datos cambiados cumplan con el tipo de dato aceptado y todos los campos estén completos, en caso contrario mandará una advertencia.

- **Resumen de usuarios que compraron entradas para el evento.**

Para cada evento deberá permitir consultar los usuarios y la cantidad total de personas que compraron entradas para un evento.
Los administradores acceden a esta función consultando primero el evento y luego las personas registradas para el mismo.
Requerimientos funcionales:

- Permitir consultas completas o parciales de la cantidad de personas que compraron su boleta para el evento.

Para el módulo de usuarios se deben tener las siguientes funcionalidades, se aclara que a pesar de que se tienen 3 roles de usuarios, las funcionalidades permanecen igual, lo único que cambia es el objeto(Cliente, Administrador o Organizador) que se almacena en las estructuras de datos.

- **Crear nuevo usuario:**

La funcionalidad deberá crear un nuevo usuario y almacenarlo en una estructura de datos.
Los visitantes deberán llenar los datos completos de un formulario para registrarse.
Requerimientos funcionales:

- El inicio de la aplicación tendrá un botón de “Sign up”, donde cada visitante nuevo podrá crear su propio usuario.
- Verificar que toda la información cumpla con el tipo de dato aceptado y todos los campos estén completos, en caso contrario mandará una advertencia.
- Comprobar que el usuario no se haya registrado anteriormente para evitar la duplicación de datos.

- **Modificar datos del usuario:**

Modificar la información de los datos del usuario asociado. Cada usuario podrá actualizar sus datos personales por medio de un formulario similar al de registrarse.

Requerimientos funcionales:
-Verificar que los datos cambiados cumplan con el tipo de dato aceptado y todos los campos estén completos, en caso contrario mandará una advertencia.

- **Eliminar usuario:**

Los usuarios podrán eliminar su cuenta en cualquier momento por medio del botón de configuración y eliminar cuenta.

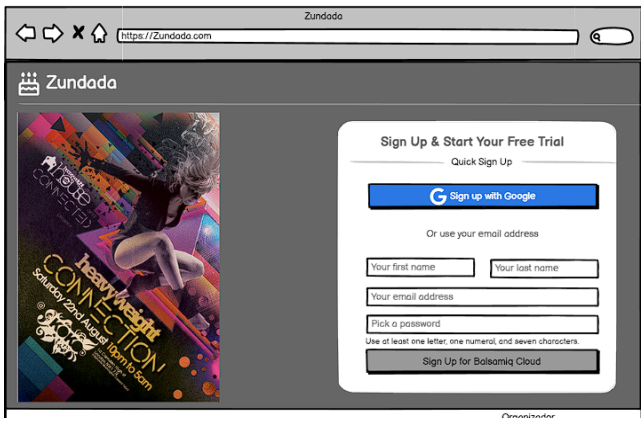
Requerimientos funcionales:
- Arrojar un mensaje de advertencia que pida la confirmación de eliminar la cuenta.

- **Consultar de entradas compradas:**

Por medio de un menú, los usuarios consultan toda la información de los eventos a los cuales compraron entradas.
Requerimientos funcionales:
- Ordenamiento por orden de eventos más recientes o por fecha de realización.

V. DESCRIPCIÓN DE LA INTERFAZ DE USUARIO PRELIMINAR

La interfaz preliminar de usuario incluye una sección de inicio con eventos destacados, además una página de inicio de sesión/registro, para que el usuario ingrese con su respectivo usuario, correo y contraseña, también incluye los de detalles del evento con información del organizador y precios de boletas y una parte de compra de boletos, además una sección para buscar más eventos



VI. ENTORNOS DE DESARROLLO Y DE OPERACIÓN

El entorno de desarrollo de "Zundada" estará configurado de la siguiente manera, el software se desarrollará principalmente en TypeScript, aprovechando su fuerte tipado y ventajas en el desarrollo web.2. Visual Studio Code será la principal herramienta de desarrollo, proporcionando un entorno altamente configurable y optimizado para TypeScript., además, GitHub será la plataforma central para el control de versiones y colaboración en equipo, permitiendo un seguimiento preciso de los cambios y una gestión efectiva del proyecto.

El software "Zundada" estará diseñado para funcionar en un entorno de producción en línea, pero no requerirá una base de datos, ya que se basará en el almacenamiento de datos en tiempo real u otras soluciones de almacenamiento de datos sin bases de datos tradicionales.

VII. PROTOTIPO DE SOFTWARE INICIAL

Este prototipo inicial se compone se maneja por medio de la consola y presenta dos módulos, la gestión de los usuarios y la gestión básica de eventos.

Link GitHub: https://github.com/Lapd1103/Zundada

VIII. IMPLEMENTACIÓN Y APLICACIÓN DE LAS ESTRUCTURAS DE DATOS

- **Array:** El array implementado en el prototipo de software es una estructura de datos que se utiliza para almacenar y gestionar la información de los usuarios. Este array contribuye a la funcionalidad del prototipo de software permitiendo realizar las siguientes operaciones:

  - Creación de Usuarios: El array permite crear y almacenar nuevos usuarios con sus nombres de usuario, correos electrónicos y claves.
  - Inserción de Datos: A través del método "crearNuevoUsuario," se insertan los datos de un usuario en el array.
  - Actualización de Datos: Se pueden actualizar los datos de un usuario específico en el array utilizando el método "actualizarUsuario."
  - Eliminación de Datos: El array permite eliminar la información de un usuario específico mediante el método "eliminarUsuario."

- Búsqueda de Datos: Es posible buscar un usuario por su nombre de usuario utilizando el método "buscarUsuarioPorNombre."
- Consulta de Todos los Datos: Se pueden consultar todos los usuarios almacenados en el array utilizando el método "consultarUsuarios."
- Almacenamiento de Datos: El array almacena de manera persistente la información de los usuarios dentro del prototipo de software.

- **Cola:**

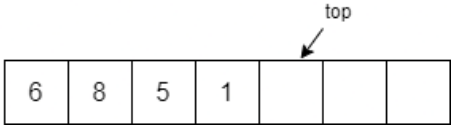
La cola se crea para almacenar al usuario por medio de su identificador único nombre de usuario. Se utilizan métodos cómo push para ingresar datos en las pruebas de casos. Además, para poder realizar las operaciones funcionales, se utiliza el find para buscar un usuario determinado que debe ser existente.

Posteriormente, si el nombre del usuario existe, se pueden consultar los datos actuales, y actualizar uno en específico. En este proceso, elimina el dato anterior, ingresa el nuevo dato y almacenarlo.

El código tiene una función main y funciones para actualizar cada tipo de dato. La función main tiene un menú de opciones que permite elegir un caso u otro por medio del switch. Además, las otras funciones tienen condiciones de límite de caracteres, si el dato se repite en la cola, entre otras, para que el usuario ingrese datos lógicos. Finalmente, usa prompt que le hace una pregunta o le da una advertencia al usuario y le permite ingresar una string o respuesta a la pregunta.

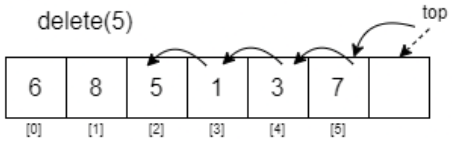
- **Pila:**

Para la implementación de la pila se utilizó un array como almacén de objetos y la variable de top que indica el índice final del array, para que de esa manera los métodos de push y pop tuvieran tiempo constante O(1).



Adicionalmente como en el programa se requiere utilizar de acciones como consultar, editar o eliminar un elemento intermedio dentro del arreglo, se implementó el método find(), que por medio de un recorrido por todo el arreglo, retorna el índice del array en el que se encuentra el objeto que necesito en concreto y de acuerdo a la acción realizó lo siguiente:

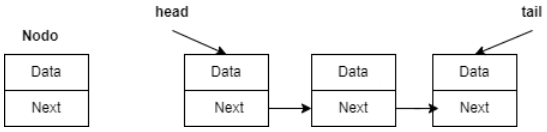
- consultar: retornar el valor del array almacenado en el índice encontrado.
- actualizar: se asigna el nuevo dato a array[i], en donde i representa la posición del dato a actualizar.
- eliminar: a partir de la posicion i, comienzo a mover cada uno de los elementos siguientes del array hasta llegar al top.



Debido a que la pila almacena los elementos según su orden de llegada, esta se implementó principalmente para el almacenamiento y gestión de los eventos, cumpliendo así con uno de los requisitos funcionales señalados.

- **Lista simplemente enlazada:**

Para realizar la implementación de la lista enlazada fue necesario implementar la clase Nodo, que es aquella que almacena los valores y además tiene un indicador (next) que señala el siguiente nodo.
Luego en la clase de lista enlazada se tienen dos indicadores adicionales, por un lado el head que me indica el nodo del inicio y el tail que indica el nodo al final de la lista.



Por último, se implementaron las funciones básicas de pushFront() y pushBack() para añadir un elemento a la lista y como método principal de consulta el popFront(), pero debido a que este último además de consultar tambien elimina el dato de la estructura, se crearon otros tres métodos de consulta:

- getHead(): retorna el dato almacenado en el nodo head.
- getTail(): retorna el dato almacenado en el nodo tail.
- find(dato): retorna el nodo anterior al elemento que busco, esta función se usó como base para crear el de actualización (update) y eliminación (erase), métodos necesarios para gestionar el almacenamiento de los usuarios y eventos.

Para el método de update, lo único que realizo es llamar la función de setData() al next del nodo que me retornó la función find(). Por otro lado la función erase() solo modifica el valor del next entregado por find(), de tal manera que elimina el dato de manera sencilla sin necesidad de correr todos los nodos de la lista.

La idea de la lista enlazada es utilizarla a forma de pila y cola para comparar los tiempos de compilación con las otras dos clases más sencillas.

IX. PRUEBAS DEL PROTOTIPO Y ANÁLISIS COMPARATIVO

- Módulo de Eventos:

Debido a que a la hora de almacenar eventos queremos que estos se consulten por el más reciente, se decidió implementar la estructura de pila como array y pila como lista enlazada; por la teoría, se sabe que la pila es más eficiente a la hora de consultar por su almacenamiento contiguo en memoria pero la acción de eliminar lo realiza en tiempo lineal (O(n)) al tener que correr los elementos dentro del array, mientras que las listas enlazadas para eliminar tienen una complejidad de O(1) porque solo resignan los valores next de un nodo, pero la

consulta se realiza en O(n) porque se debe recorrer la lista comenzando por el head.
Por esa razón se realizó la prueba de tiempo con ambas estructuras para las funcionalidades de añadir, consultar, y eliminar un elemento de la mitad de la estructura para ver cual implementación sería la más óptima.

	Añadir Eventos		Consultar Eventos		Eliminar evento de la mitad	
Cantidad	**Pila**	**Lista**	**Pila**	**Lista**	**Pila**	**Lista**
1	0,214	0,407	1,199	0,379	0,41	0,438
10	0,332	0,454	6,356	5,587	0,27	0,477
100	0,683	0,421	15,199	31,461	0,373	0,771
1000	0,913	1,714	87,535	73,737	0,414	1,153
10000	5,81	7,939	437,284	443,086	2,068	4,833
100000	19,546	11,234	2477	2784	10,098	13,697
1000000	208,334	550,469	26482	26338	13,015	19,78
10000000	3614	18776	282689	297747	860,775	1642
100000000	Error mem.	Error mem.	Error mem.	Error mem.	Error mem.	Error mem.

Tabla con los tiempos de distintas operaciones para gestionar la clase evento.

De la anterior tabla se calculó tiempo total demoraba cada estructura de datos, obteniendo lo siguiente:

	Tiempo total [ms]	
Cantidad datos	**Pila**	**Lista enlazada**
1	1,823	1,224
10	6,958	6,518
100	16,255	32,653
1000	88,862	76,604
10000	445,162	455,858
100000	2506,644	2808,931
1000000	26703,349	26908,249
10000000	287163,775	318165
100000000	Error memoria	Error memoria

De la cual se obtuvo la gráfica final, mostrada a continuación:

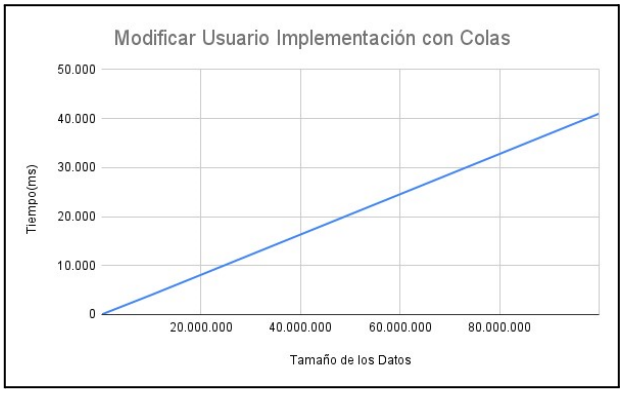
A pesar de que ambas estructuras al final tienen una complejidad amortizada lineal, es visible como el manejo de los datos toma menor tiempo cuando se realiza por medio de una pila.
La diferencia de tiempo se puede explicar si observamos los tiempos de eliminación, ya que si bien aunque la lista realiza la eliminación de tiempo constante O(1), cuando debe encontrar el elemento que quiere eliminar debe recorrer toda la lista, algo que toma más tiempo que recorrer todo un arreglo.

- Modulo de Usuarios:
  - Añadir Usuario:La implementación con una lista enlazada suele ser más eficiente en términos de inserción y eliminación de elementos, ya que estas operaciones pueden realizarse en tiempo constante O(1) si tenemos acceso directo al nodo donde se encuentra el elemento. Sin embargo, en la búsqueda de un elemento específico, la complejidad es lineal O(n) en el peor caso, ya que es necesario recorrer la lista. Por otro lado, la implementación con un array es eficiente en la búsqueda si se conoce el índice del elemento, lo que se hace en tiempo constante O(1). Sin embargo, las inserciones y eliminaciones pueden ser costosas, ya que pueden requerir reorganizar elementos en el array, lo que resulta en una complejidad de O(n).

FUNCIONALIDAD : Añadir Usuario		**FUNCIONALIDAD :Añadir Usuario**	
Implementación con Array		**Implementación con Lista enlazada**	
Tamaño de Datos	**Tiempo (ms)**	**Tamaño de Datos**	**Tiempo (ms)**
1	0,5 ms	1	0,04 ms
10	0,8 ms	10	0,08 ms
100	5 ms	100	0,16 ms
1000	50 ms	1000	0,36 ms
10000	480 ms	10000	34,75 ms
100000	800 ms	100000	163,78 ms
1000000	5000 ms	1000000	3100,16 ms
10000000	60000 ms	10000000	42000 ms
100000000	240000 ms	100000000	180000 ms

- Modificar Usuario: El código de modificar usuario tiene una complejidad en el factor que el usuario tiene el poder de decidir qué dato va actualizar. Debido a ello, se tuvo en cuenta el tiempo en el cual se puede tardar en correr el código actualizando el dato de correo electrónico, que es uno de los que tiene una mayor cantidad de requerimientos.

FUNCIONALIDAD : Modificar Usuario	
Implementación con Colas	
Tamaño de Datos	**Tiempo (ms)**
10,000	4ms
100,000	40ms
1,000,000	410 ms
10,000,000	4000 ms (4s)
100,000,000	41000 ms (41s)



Como se observa al final, la implementación para esta funcionalidad tiene un valor de O(1) ya que a pesar de que se debe buscar en todo el array la posición del dato que quiero actualizar, al estar usando un array se ahorra mucho tiempo por el manejo de espacios contiguos de memoria.

X. ROLES Y ACTIVIDADES

INTEGRANTE	ROL(ES)	ACTIVIDADES REALIZADAS
Laura Alejandra Paez Daza	Líder Coordinadora	-Planeación de las reuniones del equipo. -Distribuir las actividades pendientes. -Gestionar la distribución de documentos de la primera entrega.
Marlon David Pabon Muñoz	Animador Técnico	-Apoyo emocional al equipo. -Coordinador de pruebas técnicas del código.
María Paula Román Arévalo	Investigador Observador	- Identificación de los usuarios que utilizan la plataforma - Crear la funcionalidad para modificar los datos del usuario

XI. DIFICULTADES Y LECCIONES APRENDIDAS

Mencione las dificultades encontradas durante el desarrollo del proyecto. Además, haga alusión a las principales lecciones aprendidas durante el proceso.

- A la hora de desarrollar el esquema general del programa, fue un poco difícil establecer las relaciones entre cada una de las clases y establecer el comportamiento general del programa al tener tres roles de usuarios, pero todo se pudo esclarecer tras realizar diagramas UML.
- Dificultad en las pruebas de datos teniendo en cuenta que al modificar usuario se le da libertad al usuario de elegir qué modificar. Así que toca designar la tarea más extensa, cómo verificar correo, para tener un tiempo para la prueba de datos válida.
- Dificultad en aplicar la interacción con el usuario, debido a que se debía crear un menú en el cual el pueda seleccionar el dato a modificar de su información personal.