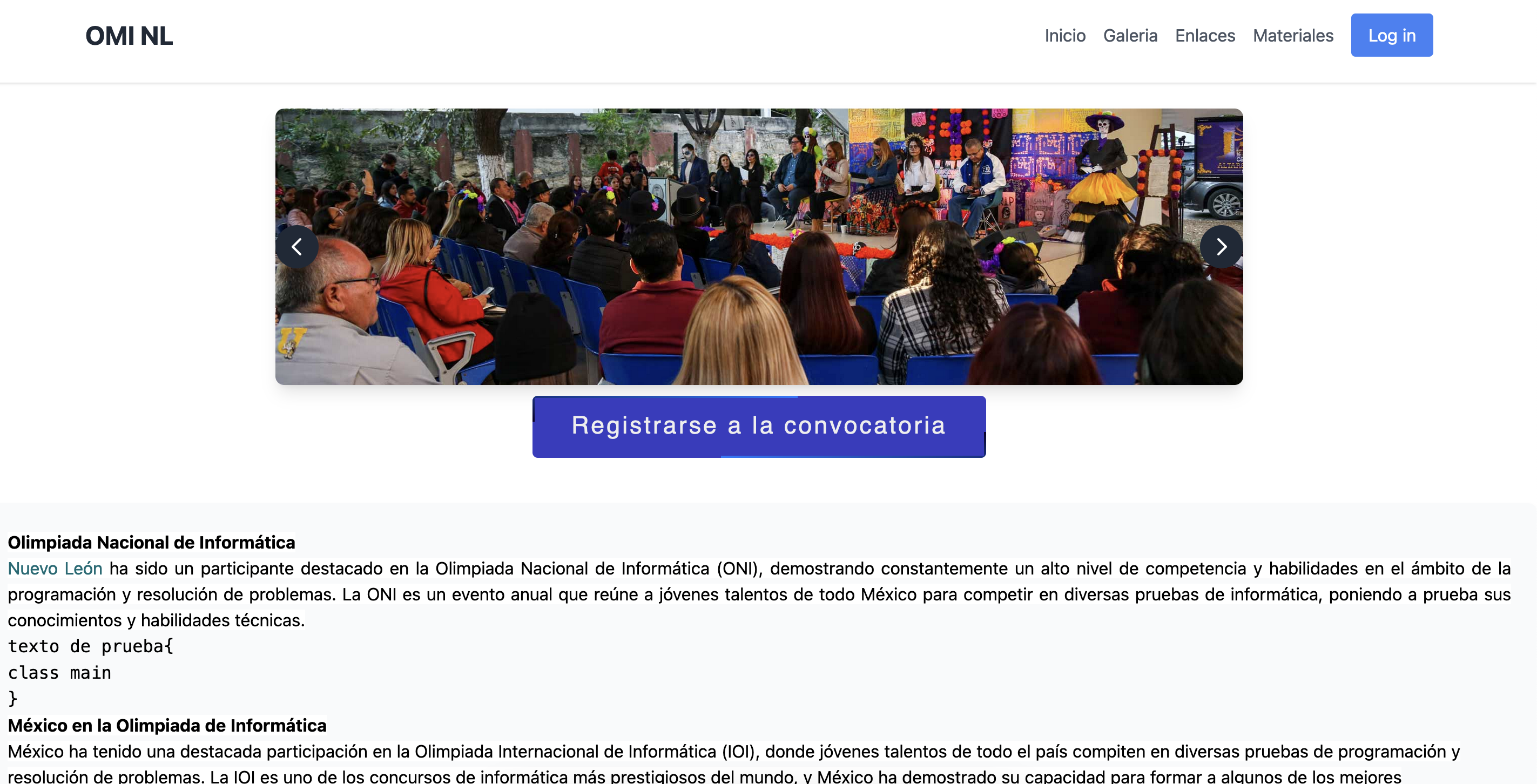
OMI NL   
Sede de la Olimpiada Estatal de Informatica  
Responsable: M.S.I. Gilberto Reyes Barrera  
Sevicio Social: Luis Fernando Flores Diaz : [fernando.floresdz@uanl.edu.mx](mailto:fernando.floresdz@uanl.edu.mx)  
  
Durante el servicio social se realizo el desarrollo de plataforma y pagina oficial para la OMI NL.   
En el presente documento se cuenta la definicion tecnica de las funcionalidades del proyecto de software.

**Funcionalidades principales**  
  
Se cuentan con 3 funcionalidades en proceso:

* Contenido de: Inicio / Enlaces / Materiales.
* Editor de contenido.
* Galeria.

Mencionaremos como se desarrollo los componentes:  
  
**Funcionalidad 1: Contenido de: Inicio / Enlaces / Materiales.**

Para los componentes de inicio, enlaces y materiales estos muestran la informacion del sitio, el objetivo de estos componentes es mostrar informacion que el administrador dedique al sitio.



El componente utiliza el servicio `ContentService` para obtener el contenido identificado por un ID. Este contenido incluye HTML que debe ser insertado en la página de manera segura. El método `getContentById()` es el encargado de realizar esta llamada al servicio, suscribiéndose a los datos proporcionados por el servicio, que devuelve una interfaz `Content` con un campo `contenido`. Una vez que se recibe el contenido, se sanitiza mediante el uso del `DomSanitizer` para evitar riesgos de inyección de código malicioso (XSS). Este HTML seguro es luego inyectado en la vista utilizando la directiva `[innerHTML]`.

**Patrones usados:**

1. Inyección de dependencias (Dependency Injection):

Este patrón se utiliza en el constructor del componente, donde se inyectan el `ContentService` y `DomSanitizer`. Esto sigue los principios de Angular, donde las dependencias se proveen a través de su sistema de inyección. En este caso:

- `ContentService` maneja las operaciones de acceso a datos (capa de servicio).

- `DomSanitizer` garantiza la seguridad del contenido HTML dinámico antes de inyectarlo en la vista.

2. Suscripción a Observables (Reactive Programming):

El método `getContentById` utiliza `subscribe()` para manejar la respuesta asíncrona que proviene del `ContentService`. Este servicio probablemente hace uso de `HttpClient` para realizar una petición HTTP. Al suscribirse, el componente puede recibir los datos una vez que estén disponibles o manejar cualquier error si la solicitud falla.

3. Seguridad de contenido con `DomSanitizer`:

La sanitización del HTML se realiza mediante el método `bypassSecurityTrustHtml()`, que es parte de las herramientas que proporciona Angular para evitar vulnerabilidades de seguridad al renderizar contenido dinámico proveniente de fuentes externas.

**Dependencias**:

1. `HttpClient` (importado en el servicio, aunque no se ve directamente en este componente): Proporciona las funcionalidades para hacer peticiones HTTP, lo cual sugiere que el contenido se obtiene desde un backend.

2. `DomSanitizer`: Ofrecido por Angular para sanitizar contenido HTML, protegiendo contra vulnerabilidades de seguridad.

3. `ContentService`: Servicio personalizado que probablemente encapsula la lógica para obtener los datos del backend, lo que sigue el patrón de separación de responsabilidades, donde la lógica de negocio o de acceso a datos no está en el componente.

Este componente se usa en una interfaz de usuario para mostrar contenido junto con otros componentes como `<app-carousel>`, `<app-convocatoria>` y `<app-donacion>`, lo que sugiere que es parte de una página que integra varias secciones dinámicas y modulares. La estructura sigue buenas prácticas de desarrollo de componentes en Angular, manteniendo la separación de responsabilidades y garantizando la seguridad del contenido dinámico.

**Funcionalidad 2: Editor de contenido**

****

**Funcionalidad principal:**

El componente `EditorInicioComponent` permite al usuario editar el contenido de una sección de la página, representada por el ID `1`. El contenido es inicialmente cargado desde el servidor utilizando el método `getContentById()`, que realiza una llamada a un servicio y vincula el contenido con la propiedad `html`. A través del editor `ngx-editor`, el contenido es editable y cualquier cambio realizado se puede confirmar mediante un botón que llama a la función `confirmContent()`, actualizando así el contenido en el servidor.

Patrones usados:

1. Inyección de dependencias (Dependency Injection):

El constructor del componente inyecta el `ContentService`, siguiendo el patrón de inyección de dependencias común en Angular, donde los servicios son proporcionados al componente. Este servicio encapsula la lógica de acceso a los datos para obtener y actualizar el contenido.

2. Ciclo de vida de Angular (`OnInit` y `OnDestroy`):

- `ngOnInit`: Se inicializa el editor (`Editor`) y se carga el contenido existente llamando al método `getContentById()`. El ciclo de vida `OnInit` asegura que estas operaciones ocurren después de que el componente está listo.

- `ngOnDestroy`: Se llama a este método para liberar los recursos del editor cuando el componente es destruido, evitando posibles fugas de memoria.

3. Suscripción a Observables (Programación Reactiva):

El método `getContentById()` utiliza `subscribe()` para manejar la respuesta asíncrona del `ContentService`. El servicio probablemente hace una llamada HTTP y devuelve un `Observable`, lo que permite al componente reaccionar cuando los datos están disponibles o manejar errores si la solicitud falla.

4. Data Binding bidireccional (`ngModel`):

El editor `ngx-editor` está vinculado a la propiedad `html` mediante la directiva `[(ngModel)]`. Este data binding bidireccional asegura que cualquier cambio realizado en el editor se refleje automáticamente en la propiedad `html` del componente, y viceversa.

Dependencias:

1. `ngx-editor`: Este es un editor de texto enriquecido que permite al usuario interactuar con contenido HTML en una interfaz gráfica. El componente utiliza tanto el menú de herramientas del editor (`ngx-editor-menu`) como el propio editor (`ngx-editor`) para proporcionar una experiencia completa de edición de texto.

2. `ContentService`: Este servicio maneja las operaciones de acceso a los datos del contenido, como la obtención (`getContentById()`) y actualización (`updateContent()`). Su uso centraliza la lógica de comunicación con el backend, separando esta responsabilidad del componente.

3. `Editor` y `Toolbar` de `ngx-editor`:

- `Editor`: Es la instancia del editor de texto enriquecido que se inicializa y destruye en el ciclo de vida del componente.

- `Toolbar`: Define las herramientas que el usuario puede utilizar dentro del editor, como negrita, cursiva, listas, enlaces, imágenes, entre otros.

Flujo de trabajo:

1. Inicialización del editor: Cuando el componente es inicializado (`ngOnInit`), se crea una instancia del editor y se carga el contenido existente desde el servidor.

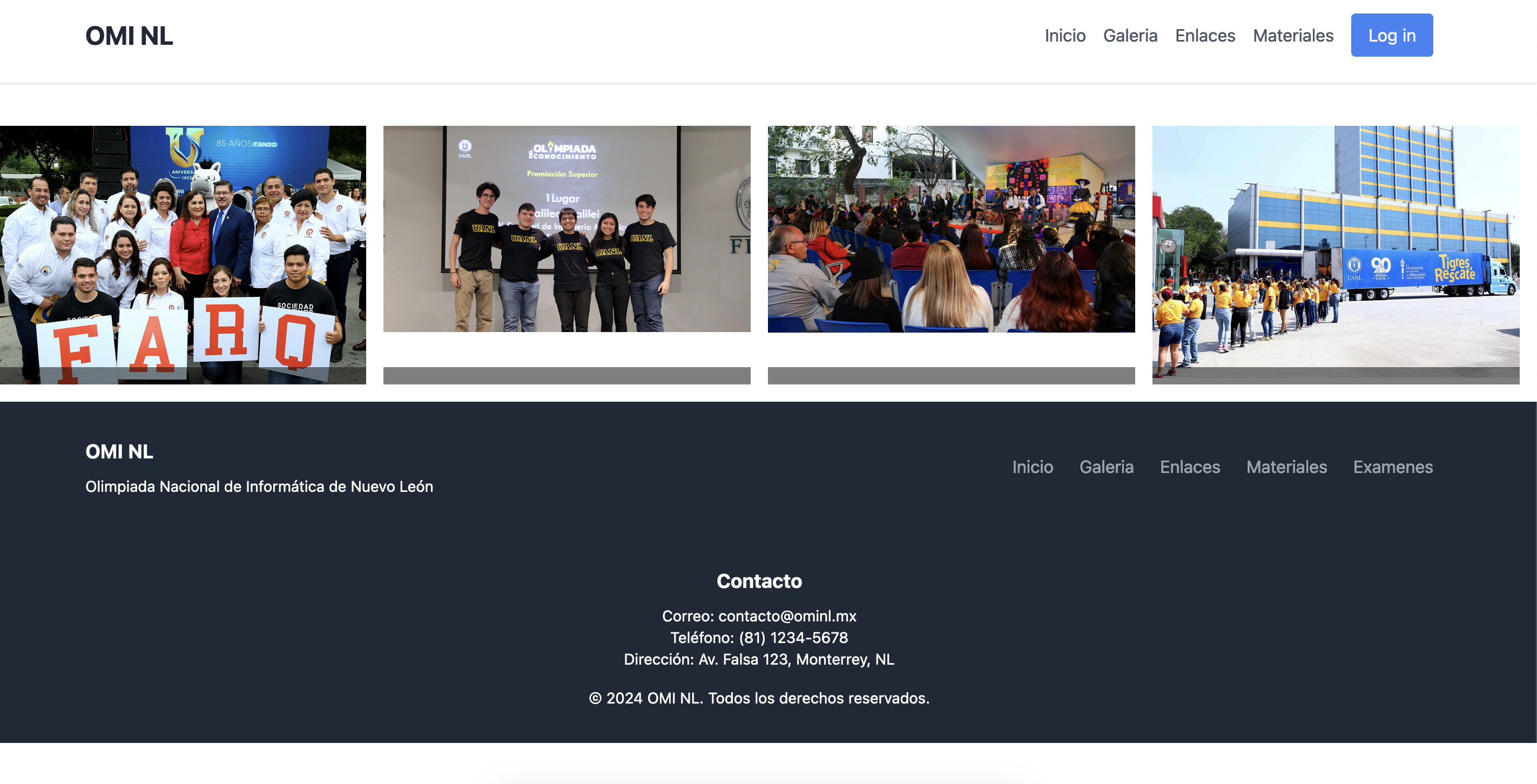
2. Edición del contenido: El usuario edita el contenido en el editor proporcionado por `ngx-editor`, y los cambios se reflejan automáticamente en la propiedad `html` del componente mediante `ngModel`.

3. Confirmación del contenido: Al hacer clic en el botón "Confirmar", el método `confirmContent()` guarda los cambios actualizando el contenido en el servidor mediante `ContentService`.

4. Limpieza de recursos: Al destruir el componente (`ngOnDestroy`), se destruye el editor para liberar recursos.

Este componente sigue buenas prácticas de Angular al separar la lógica de negocio en servicios, gestionar el ciclo de vida de los componentes correctamente, y proporcionar una experiencia de edición de contenido rica y segura para los usuarios.

**Funcionalidad 3: Galeria.**



**Funcionalidad principal**: El componente se inicializa llamando al método loadImages() en el ciclo de vida ngOnInit. Este método obtiene un conjunto de URLs de imágenes del servicio ImagesService, extrae los nombres de los archivos de las URLs y luego utiliza el método getSignedUrls() para obtener URLs firmadas de esas imágenes. El contenido de las imágenes, una vez recuperado, es asignado a la variable images, que luego se recorre en la plantilla HTML para renderizar cada imagen junto con su respectiva descripción (caption).

**Patrones utilizados**:

1. **Inyección de dependencias**: El componente sigue el patrón de inyección de dependencias, donde el servicio ImagesService se inyecta en el constructor del componente. Este servicio encapsula la lógica de obtener las imágenes desde el backend, manteniendo la separación de responsabilidades entre la lógica de negocio y la presentación.
2. **Ciclo de vida de Angular**: El componente implementa el método ngOnInit del ciclo de vida de Angular para ejecutar código una vez que el componente se inicializa. En este caso, se utiliza para cargar las imágenes inmediatamente después de que el componente está listo.
3. **Programación reactiva con forkJoin y Observables**: El método getSignedUrls() utiliza forkJoin, que es una función de programación reactiva de RxJS que permite combinar múltiples Observables y esperar a que todas las solicitudes se completen antes de continuar. En este caso, se utiliza para obtener múltiples URLs firmadas para las imágenes en paralelo, lo que optimiza el proceso de carga de imágenes.
4. **Data binding**: Las imágenes se muestran utilizando el data binding de Angular en la plantilla. La directiva \*ngFor se utiliza para iterar sobre la lista de imágenes y renderizar cada una de ellas dinámicamente. Además, las propiedades de la imagen, como src, alt, y caption, se enlazan a las propiedades de cada objeto de la lista images.

**Dependencias**:

* **ImagesService**: Este servicio se encarga de obtener las URLs de las imágenes desde una fuente externa, probablemente un backend o una API. Tiene dos funciones principales:
  1. **getImages()**: Devuelve las URLs de las imágenes.
  2. **getSignedUrl(path)**: Obtiene una URL firmada para un archivo dado, asegurando que las imágenes se puedan acceder de forma segura a través de URLs temporales.
* **forkJoin**: Es una función de RxJS utilizada para ejecutar múltiples observables en paralelo. En este caso, se utiliza para gestionar la obtención de múltiples URLs firmadas de las imágenes de manera asíncrona y eficiente.

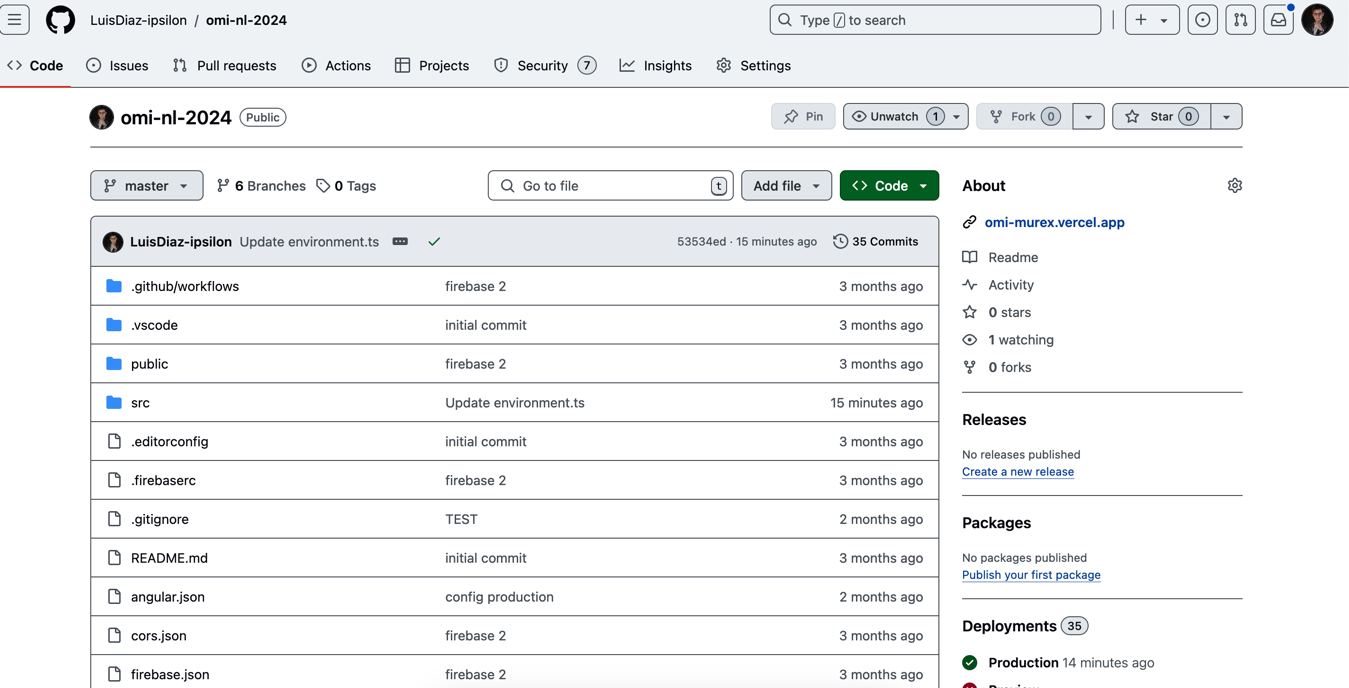
**Flujo de trabajo**:

1. **Carga de URLs de imágenes**: Cuando el componente es inicializado, se llama al método loadImages(), el cual utiliza ImagesService para obtener las URLs de las imágenes.
2. **Obtención de URLs firmadas**: Las URLs de las imágenes se procesan para extraer los nombres de los archivos, y luego se llama a getSignedUrls() para obtener URLs firmadas seguras a través de llamadas al servicio. La función forkJoin se asegura de que todas las URLs firmadas se obtengan en paralelo antes de continuar.
3. **Renderizado de imágenes**: Una vez que se tienen las URLs firmadas, estas se asignan a la variable images, y el componente renderiza cada imagen en la plantilla HTML utilizando un diseño de cuadrícula responsivo.

Este componente sigue buenas prácticas de Angular, como la separación de responsabilidades y el uso de programación reactiva para manejar operaciones asíncronas. Además, utiliza clases de Tailwind CSS para implementar un diseño limpio y responsivo, asegurando que las imágenes se vean bien en diferentes tamaños de pantalla.

**Integración de herramienta de gestión de codigo**

Se decidio usar un repositorio en github para colaborar ademas de la herramienta Trello para tener el el avance en tiempo real de todas las actividades pendientes:

 <https://github.com/LuisDiaz-ipsilon/omi-nl-2024>

Herramienta de despliege de desarrollo en pruebas:

