**UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO**

**FACULTAD DE INGENIERÍA**

# Informe de servicio social del programa colaboración en el desarrollo de proyectos médicos universitarios

**TESINA**

Que para obtener el título de

**Ingeniero en Computación**

**P R E S E N T A**

Josue Yafte Ramírez Viramontes

**DIRECTORA DE TESINA**

Mtra. Argelia Rosales Vega

**Ciudad Universitaria, Cd. Mx., 2025**

*Agradezco a mi madre y a mi hermana, ya que son mis estribos, mi aliento y mi motivo de seguir adelante. Su presencia es imprescindible en mis metas y logros.*

*A mi padre y abuelo, cuyos recuerdos vivos prevalecen como llama ardiente en mi corazón y que cada año me conmueven a ser justo de corazón.*

*A mis tíos Alfredo y Susana, que extendieron su brazo en los puntos más críticos de mi vida y son unos padres para mí.*

*A mis abuelas Manuela y Josefina, que me apoyaron en todo momento como un hijo y gracias a ellas obtuve las fuerzas para aspirar a lo más alto.*

*A mi novia Rebeca, que me motivó a tener valor y dedicación para lograr mi meta. Su apoyo y cariño es un pilar esencial en mi vida.*

*A mi mejor amigo Aaron y a mi primo Samuel, que son como unos hermanos para mí e incondicionalmente siempre estuvieron presentes.*

*A mis demás amigos, familiares y compañeros, cuya ayuda fue esencial para mi desarrollo profesional.*

*A mis profesores, que me otorgaron el conocimiento necesario, me enseñaron a aprender y fueron indispensables para llegar a este punto.*

*A mi responsable Argelia, que su gracia, apoyo y conocimiento siempre estuvo disponible para mí permitiendo que sea posible lograr esta meta.*

# Índice


Introducción ..... 5

Planteamiento del problema / Justificación ..... 6

Descripción del proyecto ..... 7

- Funcionalidad del *e-cartel* ..... 7
- Aspecto del *e-cartel* ..... 7

Análisis de requerimientos ..... 9

- Requerimientos funcionales ..... 9
- Requerimientos no funcionales ..... 10

Diseño del sistema ..... 10

- Diseño visual, maquetaciones y flujo de la aplicación web ..... 10
- Especificación de *frontend* ..... 14
  - Elección de *framework* ..... 14
  - Módulos ..... 14
  - Componentes ..... 15
  - Servicios ..... 16
- Especificación de *backend* ..... 16
  - Elección de lenguaje ..... 16
  - Métodos ..... 17
  - Base de datos ..... 17

Implementación (¿se podrá hace un esquema o diagrama del proceso de implementación?) ..... 17

- Creación de *frontend* con *Angular* ..... 18
  - Implementación de *Routing module* y *Lazy loading* ..... 20
  - Implementación de página principal en *Home Module* ..... 21
  - Implementación de página de edición en *Design Module* ..... 22
  - Implementación de Servicios ..... 35
  - Implementación de diseño responsivo para dispositivos móviles. ..... 39
- Creación de *backend*: Base de datos y *API* ..... 43
  - Creación de Base de datos ..... 43
  - Creación de tabla carteles y *queries* ..... 43

Implementación de *API* y configuración de conexión a la base de datos ..... 44
Implementación de *endpoints* para la *API* ..... 45
Pruebas de funcionamiento ..... 49
Despliegue en servidor local *AppServ* ..... 50
Página principal y redirección a rutas sin acceso ..... 51
Página de edición para cartel ..... 53
Guardado y actualización de cartel ..... 54
Vista previa y página final del cartel ..... 56
Conclusiones ..... 61
Bibliografía ..... 63

# Introducción

La Facultad de Medicina (FM) de la Universidad Nacional Autónoma de México (UNAM), por medio de la Secretaría de Educación Médica (SEM), fue la responsable de organizar el Congreso Internacional de Educación en Ciencias de la Salud (CIECS), en modalidad virtual, evento que se llevó a cabo del 5 al 9 de septiembre de 2022 con el tema "Experiencias en la educación remota de emergencia y tendencias educativas pospandemia", donde junto con el director y personas expertas en el tema exponen diferentes puntos de vista sobre el ámbito académico y educativo respecto a la pandemia iniciada en el año 2020.

El contexto anterior abrió paso a que tanto estudiantes como docentes implementaran y se adaptaran a las diferentes herramientas tecnológicas vía remota para el aprendizaje, las cuales llegaron para quedarse y aún después de la pandemia siguen siendo la mejor opción para facilitar la enseñanza, el aprendizaje y la evaluación.

El congreso dio lugar a distintas actividades académicas, una de ellas se enfocó al uso de las herramientas tecnológicas en la educación, así como experiencias que los profesionistas proponen y pusieron en práctica en pandemia. Éstas fueron presentadas por medio de un trabajo oral o un *e-cartel*. El *e-cartel* es una nueva propuesta por parte de la FM que consiste en presentar un trabajo académico con apoyo de un cartel científico de forma electrónica, es decir, el *e-cartel* podrá ser visualizado desde una página web y, además, contendrá recursos multimedia.

Dado lo anterior, en la presente tesina se documenta el desarrollo y los resultados de la implementación de una aplicación digital que es requerida para el CIECS, la cual es una herramienta que facilita a los participantes del evento la exposición de sus actividades solicitadas, además de la oportunidad de publicarlas en una página web.

La herramienta *e-cartel* es una tecnología que aún está siendo optimizada para que cada participante pueda usarla y personalizarla, por ello, en el presente documento se muestra el trabajo realizado durante el servicio social donde se propuso una solución a la problemática y dando como producto una aplicación web que sintetiza los procesos para la creación, modificación y visualización del cartel que anteriormente no había.

## Planteamiento del problema / Justificación

Si bien es cierto que durante la pandemia las herramientas digitales fueron mayormente usadas, para poder realizar el evento del CIECS de forma digital, la SEM se propuso una nueva modalidad para la presentación de los trabajos académicos aceptados: el *e-cartel*. Es importante implementarla debido a que aún no hay una herramienta digital propia que permita a los ponentes presentar su *e-cartel* científico con los requerimientos exactos que el CIECS solicita. Adicionalmente, se disminuye el uso de material impreso que posteriormente no se reutiliza lo cual también ofrece una reducción de contaminación al medio ambiente.

Ante esta situación, surgió la necesidad de desarrollar una aplicación digital con los requerimientos y recursos disponibles en la SEM. Otra ventaja que tiene esta propuesta es que permite añadir nuevos elementos que en un cartel físico no podrían tener, como son los hipervínculos o un carrusel de imágenes. A futuro, esta herramienta podrá tener un escalado significativo para añadir inclusive contenido más interactivo. Por consiguiente, esta propuesta permitirá a los ponentes redactar y presentar toda su información en un solo espacio, de forma presencial o virtual, además de que sólo necesitará un recurso físico: su computadora.

## Objetivo

Implementar una herramienta digital en formato de aplicación para página web, que facilite la creación de carteles electrónicos que se presentarán durante el Congreso Internacional de Educación en Ciencias de la Salud, de la Secretaría de Educación Médica, de la Facultad de Medicina de la UNAM.

# Descripción del proyecto

Relacionado a la introducción de este documento, el proyecto consiste en implementar una herramienta digital que permita elaborar un *e-cartel* con contenido multimedia y que pueda visualizarse como una página web. El aspecto visual queda a consideración del desarrollador, sin embargo, serán imprescindibles los formatos, estructuras y secciones que deberá contener el cartel y que podrá editar el usuario final.

## Funcionalidad del *e-cartel*

Las funcionalidades principales del *e-cartel* son la edición y texto con contenido multimedia. Una vez que el usuario tenga la versión final de su *e-cartel* se guarda y finaliza, con opción a modificarse las veces que considere necesarias. El *e-cartel* no cubre con la funcionalidad de publicarse, pero sí de finalizado ya que pertenece a un proceso interno de la página que requerirá un visto bueno, sin embargo, eso no impedirá que pueda visualizarse posteriormente.

## Aspecto del *e-cartel*

El cartel es un recurso visual con una relación aproximada de 1.5 (cm/cm) de su altura entre su ancho, requiere ciertos campos indispensables como título, introducción, metodología, objetivos, resultados, gráficas e imágenes, conclusión, referencias, datos de contacto y opcionalmente agradecimientos. Es ideal para ejemplificar una investigación académica.

Cuenta con una distribución en tres formatos diferentes, todos con un *banner* que requiere tener en el *header* del cartel para que el usuario pueda elegir, de entre ellos, el que más sea de su agrado, los formatos considerados son los que se muestran a continuación:


Imagen 1. Esquema para formato 1 del *e-cartel.*

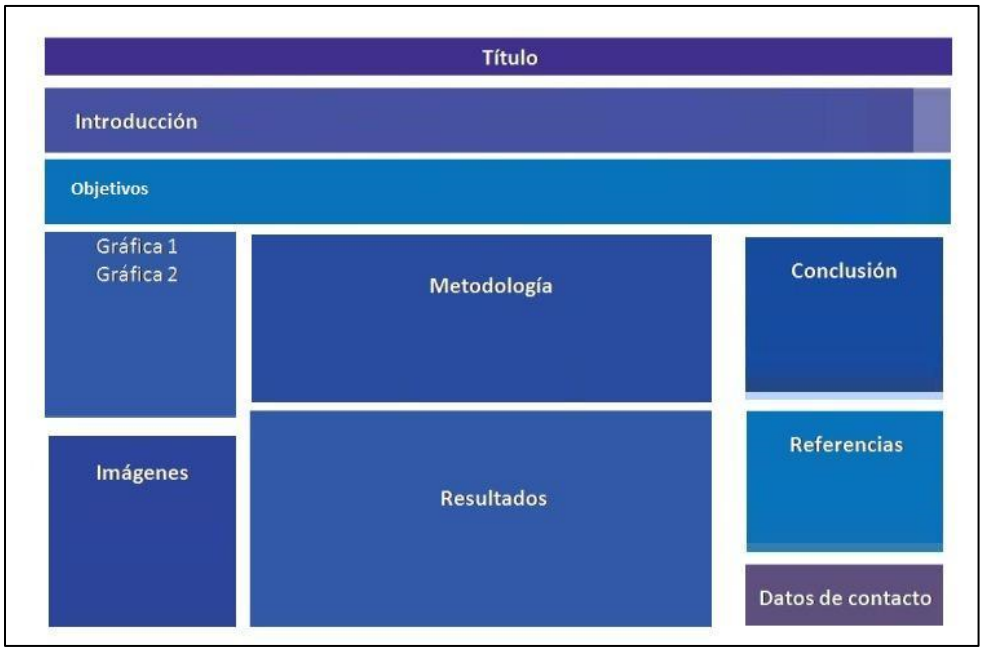



Imagen 2. Esquema para formato 2 del *e-cartel.*

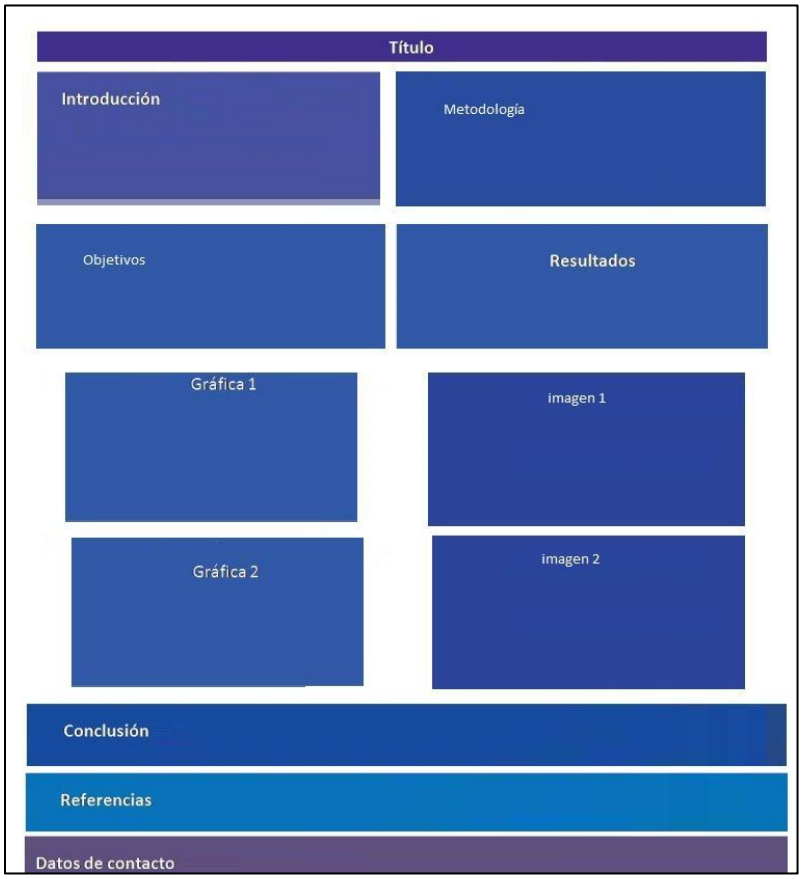



Imagen 3. Esquema para formato 2 del *e-cartel.*

# Análisis de requerimientos

Una vez definida la descripción del proyecto se consideraron los siguientes requerimientos para ~~poder~~ realizar el diseño de la herramienta y posteriormente implementarla.

## Requerimientos funcionales

- El usuario podrá editar las cajas de texto, así como añadir hipervínculos y contenido multimedia de acuerdo con la sección que desea modificar.
- El usuario podrá guardar su *e-cartel* independientemente de si está o no finalizado.
- El usuario podrá hacer una visualización previa del *e-cartel* mientras lo edita, independientemente de si está o no finalizado.

- El usuario podrá poner en estado de finalizado su *e-cartel*, así como volverlo a modificar.

### Requerimientos no funcionales

- El usuario podrá personalizar los textos, colores e imágenes del *e-cartel* con la limitación que contemplan los formatos de la Imagen 1, Imagen 2 e Imagen 3.
- El *e-cartel* contendrá un *banner* que se encontrará en el *header* del cartel y que es imprescindible.
- El usuario podrá cambiar de formato.

## Diseño del sistema

A continuación, se muestra el diseño de acuerdo con lo analizado en los requerimientos para un buen funcionamiento del sistema.

### Diseño visual, maquetaciones y flujo de la aplicación web

Inicialmente se requirió un identificador, es decir, el nombre que el usuario desee poner a la versión de su *e-cartel*. Así, de acuerdo con el flujo, debe existir una primera página que recabe el dato para guardar el cartel.

Dadas las funcionalidades y características del *e-cartel*, se necesita que la herramienta tenga un panel con las opciones de diseño: colores del cartel, guardar, visualizar y finalizar. Dicho panel estará situado como un *sidebar* colapsable del lado izquierdo para que sea fácil de ocultar y editar, como se muestra en la Imagen 4 y 5. Además, contará con un botón flotante en la esquina superior izquierda que sea visible en todo momento para ayudar al usuario a ubicar las opciones anteriores.


Imagen 4. Maquetación de la vista principal con *sidebar* visible.


Imagen 5. Maquetación de la vista principal con *sidebar* colapsado.

Por otra parte, las secciones del *sidebar* son las siguientes:

- *File*: visualización previa, guardado y finalizado (Imagen 6).
- Formato: opciones de formato (Imagen 7).
- Color: cambiar el color en general de la estructura del *e-cartel* (Imagen 8).

Adicionalmente en la parte superior hay una entrada para asignar el nombre de la versión que se trabaja, este nombre es diferente del título y servirá más como un identificador a futuro para el mismo usuario.


Imagen 6. Maquetación con botón *File* desplegado.


Imagen 7. Maquetación con botón Formato desplegado.


Imagen 8. Maquetación con botón Color desplegado.

En resumen, el flujo de la aplicación consta de una página principal que obtiene el nombre de la versión del *e-cartel,* esta redirige a una página de edición con herramientas para cambiar formato y colores, así como el guardado, vista previa y finalizado. Además, se puede visualizar el trabajo actual u otras versiones guardadas. La vista previa permitirá visualizar un *e-cartel* ya existente en la base de datos, o el que se está actualizando en ese instante.

## Especificación de *frontend*

Con base en el diseño propuesto para la página y teniendo claro el funcionamiento, se evaluaron los siguientes puntos para hacer más efectivo y amigable el proceso de desarrollo visual de la interfaz.

### Elección de *framework*

Para este proyecto fue indispensable el uso del *framework Angular* 16, ya que al ser contenido dinámico se requiere de gran abstracción para relacionar componentes con su formulario y, precisamente, *Angular* permite tener una modularidad alta para dividir la app en componentes. Además, puede crear módulos necesarios para las rutas de navegación y enlazar el código *Typescript* dentro del documento de la estructura *HTML* (*HyperText Markup Language*) para usar las variables necesarias del mismo.

Por otra parte, fue esencial el uso de *Bootstrap* ya que sus clases permiten una distribución moldeable y sencilla de usar para cualquier dispositivo. Otro recurso que también se usó es el módulo de *TinyMce* para facilitar la personalización de los campos de cada formato y tener un tamaño específico de letra, color, alineación, estilo y poder adjuntar *links*.

### Módulos

La base de la aplicación en *Angular* requiere que se parta de una *App Component,* la cual es el componente principal, de este desprenden cuatro módulos como se ilustra en la Imagen 9, cada uno con funciones diferentes. El primero es *Shared Module*, contiene elementos que todos los módulos pueden tomar, por ejemplo: la página de error. El segundo, *Design Module*, contiene elementos que sirven para el diseño de la página, por ejemplo: el panel del *sidebar* y la página editable con los formatos para el *e-cartel* y sus

respectivos *inputs.* El tercero es *Visual Module*, que contiene únicamente el *e-cartel* visual que se mostrará en el *preview* o la página final. Por último, *Home Module,* este contiene un formulario previo para poder guardar el nombre del *e-cartel* y así poder empezar la edición.

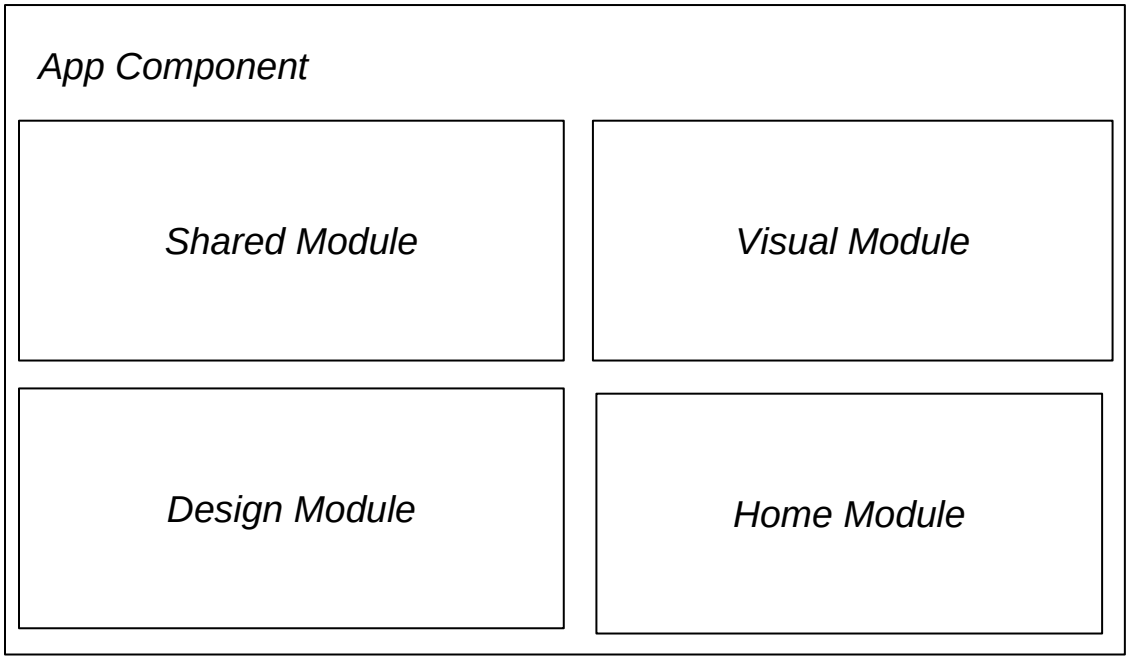



Imagen 9. Esquema general de módulos de la aplicación principal.

## Componentes

Cada módulo contiene componentes e interfaces que permiten conectar la información con su funcionalidad y, respecto a esto, se describe cada uno:

*Shared Module*: Contiene únicamente la página de error. Este componente es lanzado una vez que el *path* de la ruta en la *URL (Uniform Resource Locator)* a la que se intenta ingresar es inválido o no existe. Por otra parte, se usan otros módulos como *Http Client Module* que permite obtener los servicios para hacer consultas *post* que conectarán con la *API (Application Programming Interface)*. Además, *Editor Module* es usado para añadir cajas de texto personalizables y *Forms Module* administra los campos para convertirlos en reactivos y herramientas para validarlos. Los dos últimos son compartidos por *Shared Module* para no importarlos en cada uno.

*Visual Module*: Contiene el componente de página Solo vista, no permite editar, y sólo muestra la información de entrada de un *e-cartel* existente en la base de datos, o bien, el que se desea previsualizar.

*Design Module*: Tiene más componentes como el *layout component* que contiene el *sidebar*; y el *editable page component* que muestra los diferentes formatos del *e-cartel* para editar. Además, cuenta con los tres formatos para cambiar entre ellos la distribución de las secciones y componentes complejos. Por ejemplo: un arreglo de entrada para enlaces de imágenes que posteriormente se muestran en un carrusel.

### Servicios

*Design Module* contiene el *sidebar*, siendo el único que ofrece los servicios que controlan las funciones principales como Guardar y Finalizar, que a su vez se conectan con el *backend* por medio de una *API*. Estas se encuentran con métodos en el servicio llamado *Back service* usando el *Http Module,* que ya comparte el *Shared Module,* y permite realizar las peticiones post hacia la *API*. Por otro lado, el *Design service* ocupa servicios que conectan componentes hermanos como *layout component* del *Design Module* con los de formatos, estableciendo una comunicación donde el *layout component* emite los cambios realizados en los controles del *sidebar* (cambio de formato o de color) y los aplica en el componente de formato visual que en ese momento se está editando.

## Especificación de *backend*

La funcionalidad del *backend* se centra en solo realizar inserciones y actualizaciones dentro de la base de datos correspondiente al sitio en productivo.

### Elección de lenguaje

Para realizar los servicios de *backend*, actualizar y guardar el *e-cartel*, se creó una *API* a través del lenguaje *PHP* y con el método *post* para consumir los *request* que solicite el *frontend*. Este proceso no propiciará un problema de seguridad ya que esencialmente en productivo no estará bajo el control de esta aplicación, sino que ya existe una arquitectura con seguridad establecida para añadir estos nuevos procesos y la nueva *API* implementada solo añadirá métodos que modifiquen una sola tabla.

### Métodos

Respecto a los métodos para el diseño del *backend*, solo se necesitan dos: el método actualizar que permitirá añadir cambios o insertar y el método finalizar que permite poner una bandera de la base de datos en un estado de 0 a 1 para saber si está listo para publicarse.

### Base de datos

En la base de datos, lo que corresponde en productivo, ya está creado, lo único que se agrega es una tabla nueva que registrará cada *e-cartel* sin asociarlo con su usuario ya que esa parte queda fuera del control del mismo proyecto. La tabla contiene: el *ID (Identificador)* del *e-cartel*, el nombre con el que el usuario desea guardarlo, el cifrado en *base64*, la fecha de creación, la fecha de última actualización y si está finalizado o no, como lo muestra la Imagen 10.

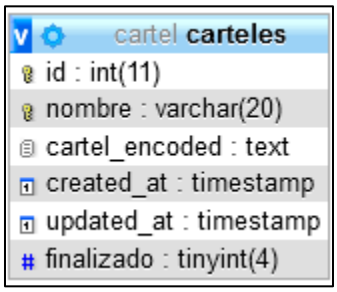



Imagen 10. Estructura de tabla de carteles dentro de la base de datos.

## Implementación

Para la implementación, primero se realizó la construcción del *frontend* en *Angular* y posteriormente el *backend* en *MySQL* y *API* en *PHP* como conexión entre el *backend* y el *frontend* de forma local (Imagen 11).

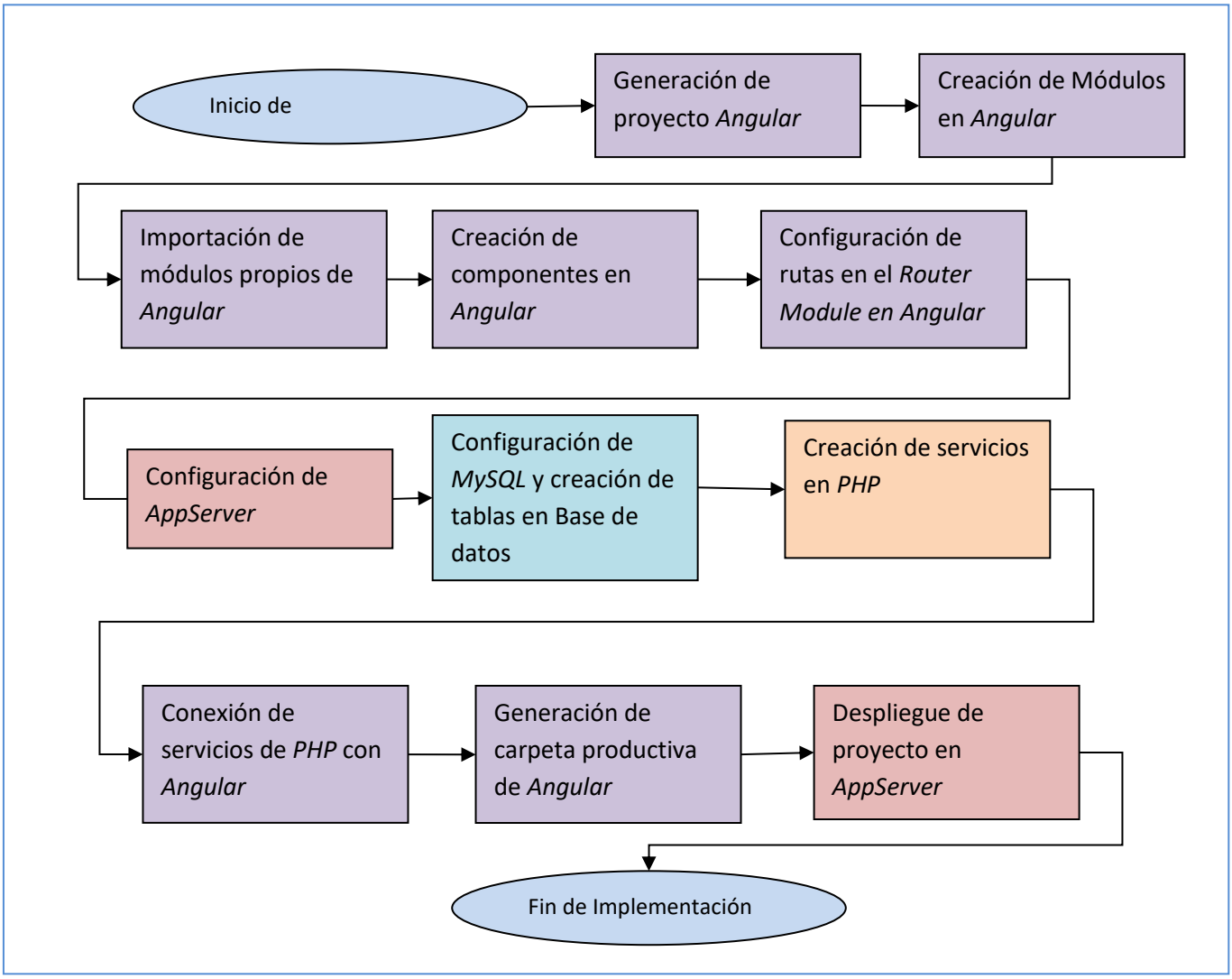



Imagen 11. Diagrama de implementación.

## Creación de *frontend* con *Angular*

Como primer paso, se realizó la creación del proyecto *e-cartel* con la herramienta *Angular CLI,* se añadió el *Routing Module* que gestiona las rutas que se usarán para la navegación dentro de la página y se configuró el uso de *CSS* para hojas de estilos como se muestra en la Imagen 12.

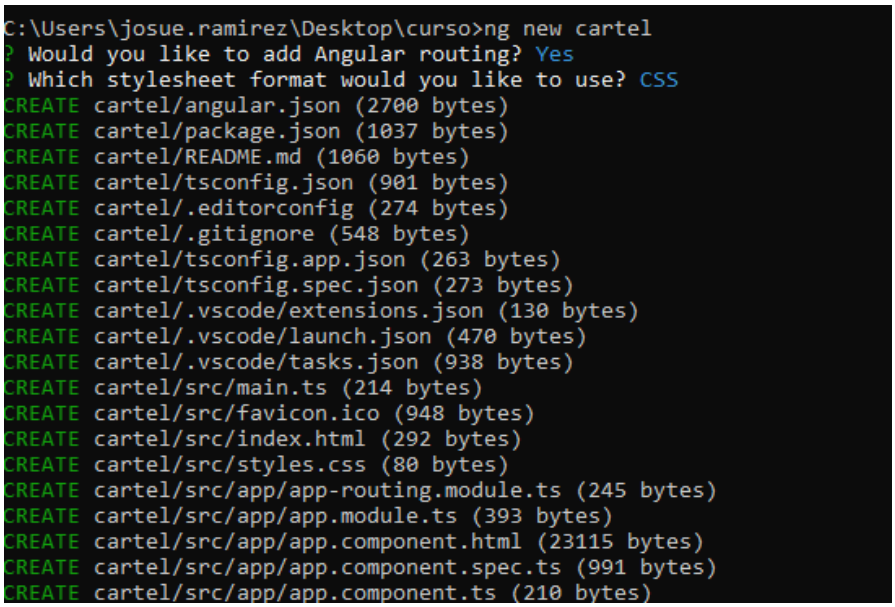



Imagen 12. Creación de proyecto inicial en *Angular*.

Posteriormente, se crearon los módulos (Imagen 13) y los componentes (Imagen 14), se añadieron los servicios *Back service y Design service* dentro de *Design Module* siendo el único que puede proveerles su respectivo servicio a los demás módulos (Imágenes 15 y 16).

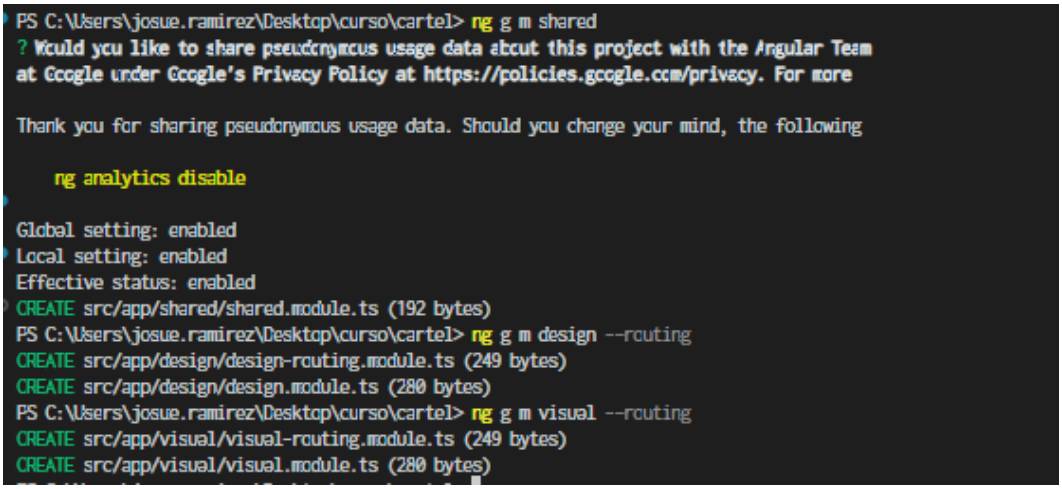



Imagen 13. Creación de módulos en *Angular*.


Imagen 14. Creación de componentes en *Angular* para *Visual Module*.

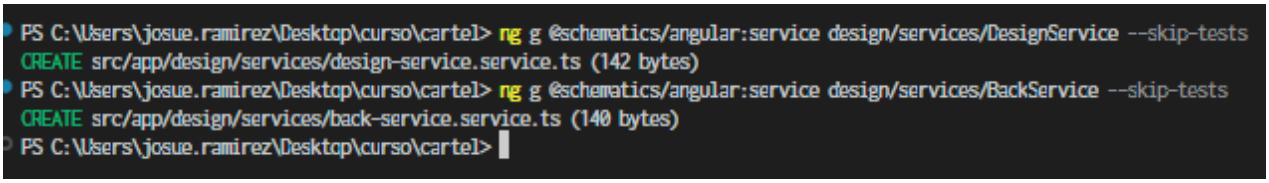



Imagen 15. Creación de servicios en el *Editor Module*.

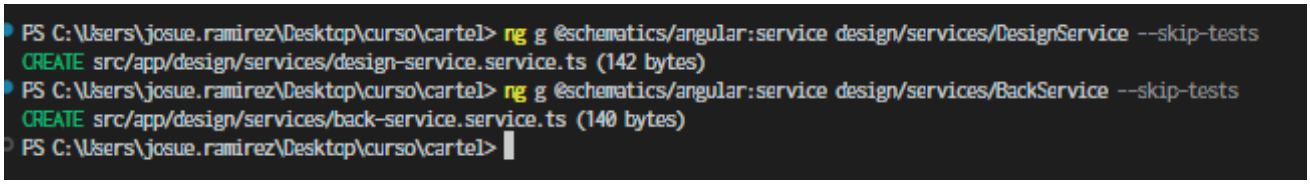



Imagen 16. Adición de *TinyMce* para el *Editor Module*.

## Implementación de *Routing module* y *Lazy loading*

Una vez creados los archivos se diseñaron las rutas de flujo y la restricción a rutas no válidas que redireccionan a una página con el error 404 que significa “no se encontró”, dicha página se comparte con *Shared Module* (Imagen 17). La ruta vacía redirige a ***/home/new-page,*** página principal donde pide el nombre de la versión antes de continuar a editar la misma versión de cartel. Para el *Visual Module* a partir de la raíz tiene la ruta ***/ver***, que a su vez tiene dos rutas hijas: ***/ver/cartel*** que muestra los *e-carteles* guardados en la base de datos; y ***/ver/preview*** que muestra el *e-cartel* que se está editando. Por

último, para el *Design Module* se tiene la ruta ***/editar/nuevo*** que permite continuar la edición del *e-cartel*.


Imagen 17. Página de error 404.

Una vez que las rutas para el flujo de la página configuradas, se añadió la librería de *Bootstrap 5.3* y los módulos complementarios para *Shared Module*, se destaca *Editor Module* porque tiene la caja de texto con hipervínculos, estilos y tamaños de letra dependiendo de los componentes que se modifican del *e-cartel*.

### Implementación de página principal en *Home Module*

Se creó la página *Get Name Page Component*, como se observa en la Imagen 18, que permite ingresar un nombre no existente en la base de datos, no sea vacío y esté entre 5 y 40 caracteres para redireccionar de la ruta ***/editar/nuevo***, de lo contrario mostrará un error.


Imagen 18. Página principal ingresando a la ruta raíz del servidor.

## Implementación de página de edición en *Design Module*

Se programó la ruta a la que redirecciona la página principal, donde se realiza la mayor parte de las acciones por parte del usuario. Para comenzar, se envió una señal a través de un tipo de dato llamado *Observable*, que permite obtener el nombre y enviarlo a un módulo que no es hijo de este, de modo que si no se ha recibido será redirigido al *home* y así se evite un mal flujo del programa y se ingrese a editar un nuevo *e-cartel* si aún no se le pone nombre.

En el componente de *layout*, página principal de diseño, se añadió un elemento llamado *offcanvas* que permite integrar un *sidebar* al que se añadieron, en forma de acordeón, las herramientas que maneja el usuario: *File*, que contiene los botones Guardar, *Preview* y Finalizar (Imagen 19); Formato, con tres *radiobuttons* para elegir el formato preferido (Imagen 20); y la sección de Color, con cinco *input*, Fondo del lienzo, Fondo del título, Fondo y texto de subtítulos y Fondo de apartados (Imagen 21). Además, hay un botón flotante en forma de flecha que permite abrir el *sidebar* y otro para cerrarlo en caso de no requerir su función.

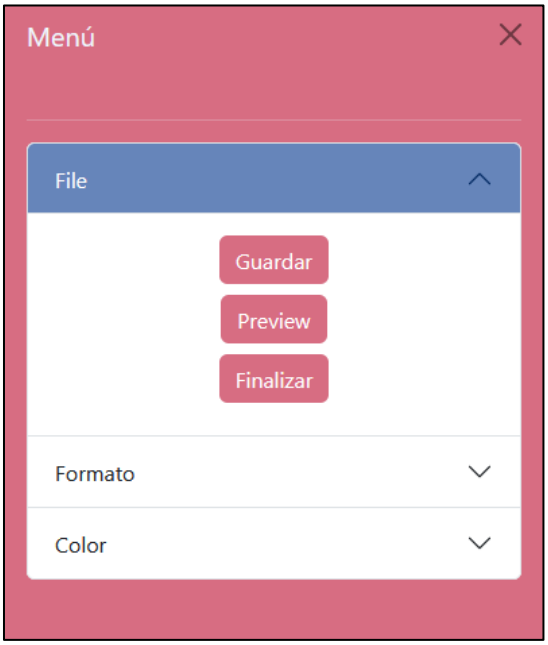



Imagen 19. *Sidebar* de la página de edición con opción *File* desplegada.

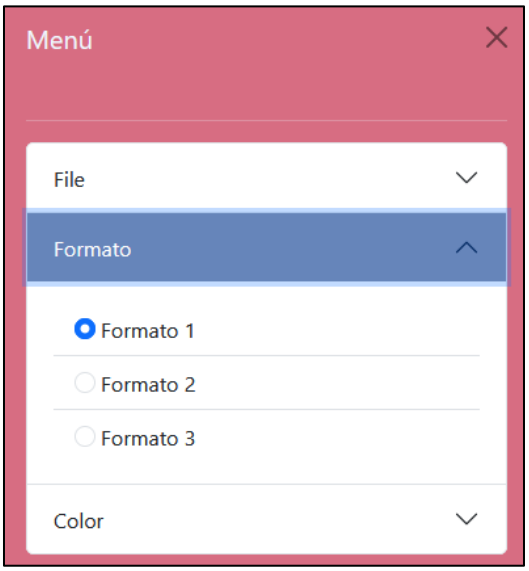



Imagen 20. *Sidebar* de la página de edición con opción Formato desplegada.

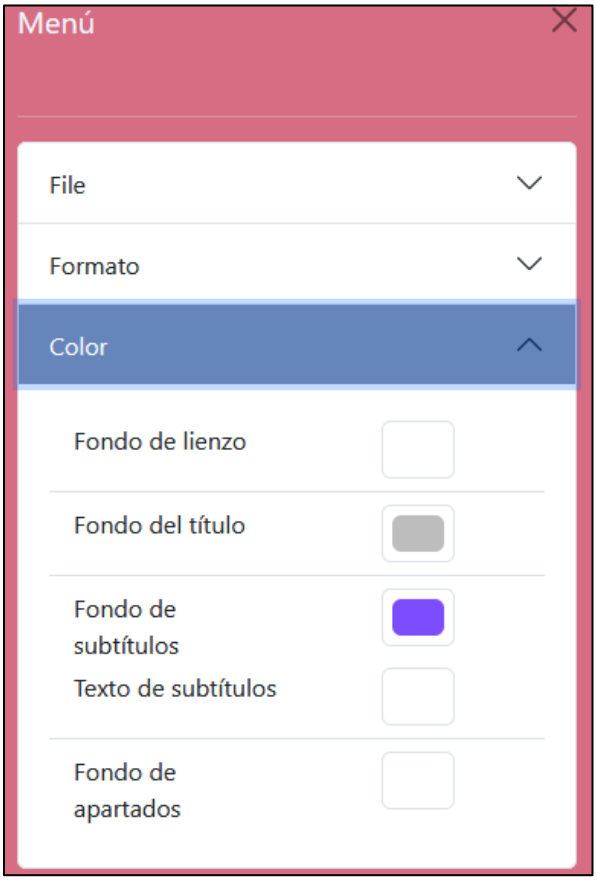



Imagen 21. *Sidebar* de la página de edición con opción Color desplegada.

Posteriormente, se realizó un *label* que muestra el nombre asignado por el usuario e identifica qué *e-cartel* está editando, se dibujó el *header* (una imagen de fondo) que corresponde al logo del congreso y, en tercera instancia, se puso una caja de texto con el *Editor TinyMce* en la cual el usuario escribirá su tema (Imagen 22). Lo anterior se agregó en el *Layout Component* perteneciente al *Design Module* ya que son elementos estáticos y lo único que cambia, de acuerdo con el reactivo de los *radiobuttons* del *sidebar,* es el contenido de los formatos.

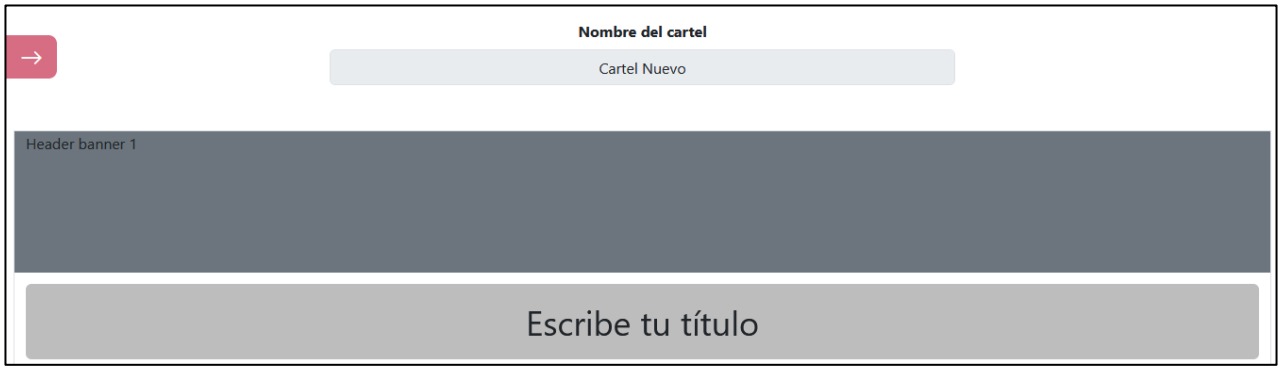



Imagen 22. Elementos estáticos en Página de edición para todos los formatos y *Sidebar* colapsado.

Cada formato es un *template* que se creó en un componente compartido con la información que el usuario edita. Cada apartado cuenta con una caja del editor *TinyMce* para personalizar al dar clic sobre el texto inicial como se ilustra en la Imagen 23.

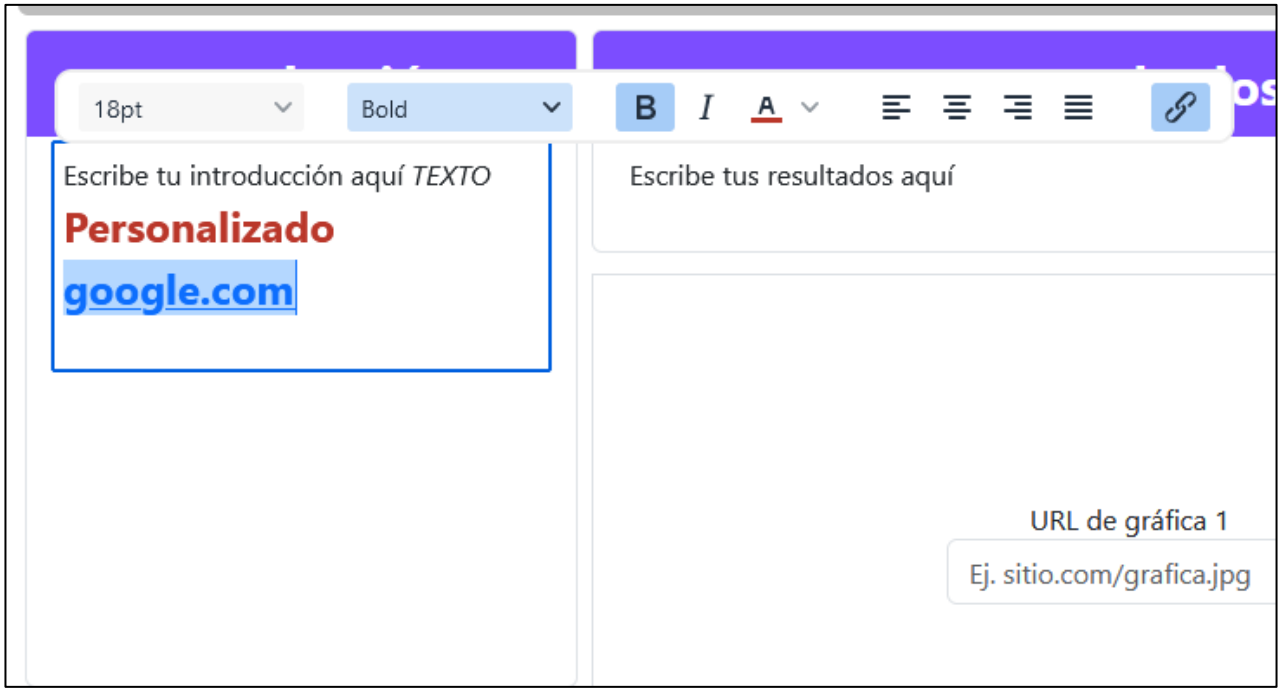



Imagen 23. Edición de texto en apartado Introducción del formato 1 con *TinyMce*.

De la misma forma, se realizó un *input text* que solo permite ingresar un enlace a una imagen de internet que permita ilustrar sus gráficas. Por otro lado, se creó un componente

en la sección de *tools* para el *Design Module*, el cual integra un *input* y un botón para añadir varios enlaces que posteriormente se usarán en un elemento llamado Carrusel de imágenes. Este a su vez, también permite eliminarlos cuando el usuario ya no desea mostrarlos (Imagen 24).

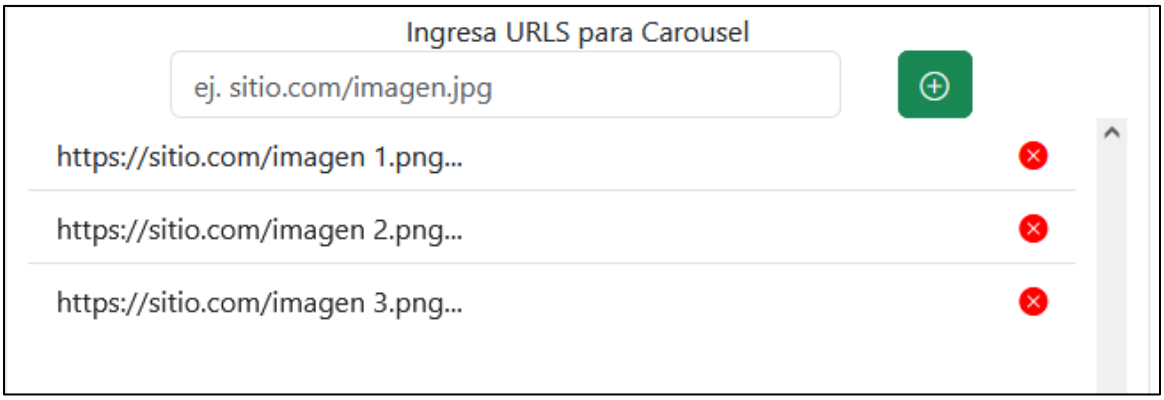



Imagen 24. Componente *Input* Carrusel con enlaces de ejemplo añadidos.

Se realizó la función de Cambio de formato para que los datos puedan ser compartidos entre los tres formatos y, al cambiar entre estos, se siga mostrando la misma información. En la estructura variable, se optó por integrar herencia, siendo el *Editable Page Component* el padre de *Format One Component*, *Format Two Component* y *Format Three Component.*

Los componentes unitarios anteriores fueron distribuidos de diferente forma entre cada uno de los formatos de acuerdo con su plantilla y heredando la información que le corresponde. Cada formato fue estructurado por columnas de tamaño variable respecto al ancho de la página visible como lo permite *Bootstrap*. Por otra parte, el *e-cartel* contiene un tamaño vertical mínimo que conforme el usuario edita y llena, cada apartado va creciendo, dependiendo de la cantidad de palabras que el usuario añada. Las siguientes imágenes ilustran el resultado de la implementación de los elementos de entrada distribuidos en cada formato.


Imagen 25. Primera parte del formato 1 editable.

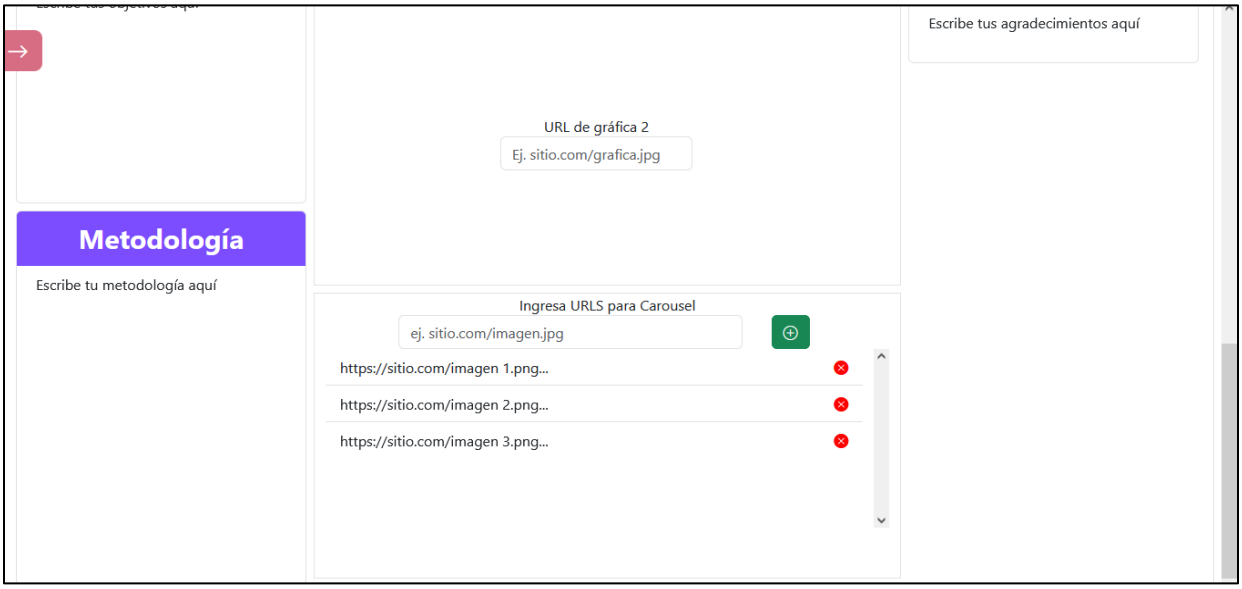



Imagen 26. Segunda parte del formato 1 editable con *input* para añadir varias imágenes.


Imagen 27. Primera parte del formato 2 editable.

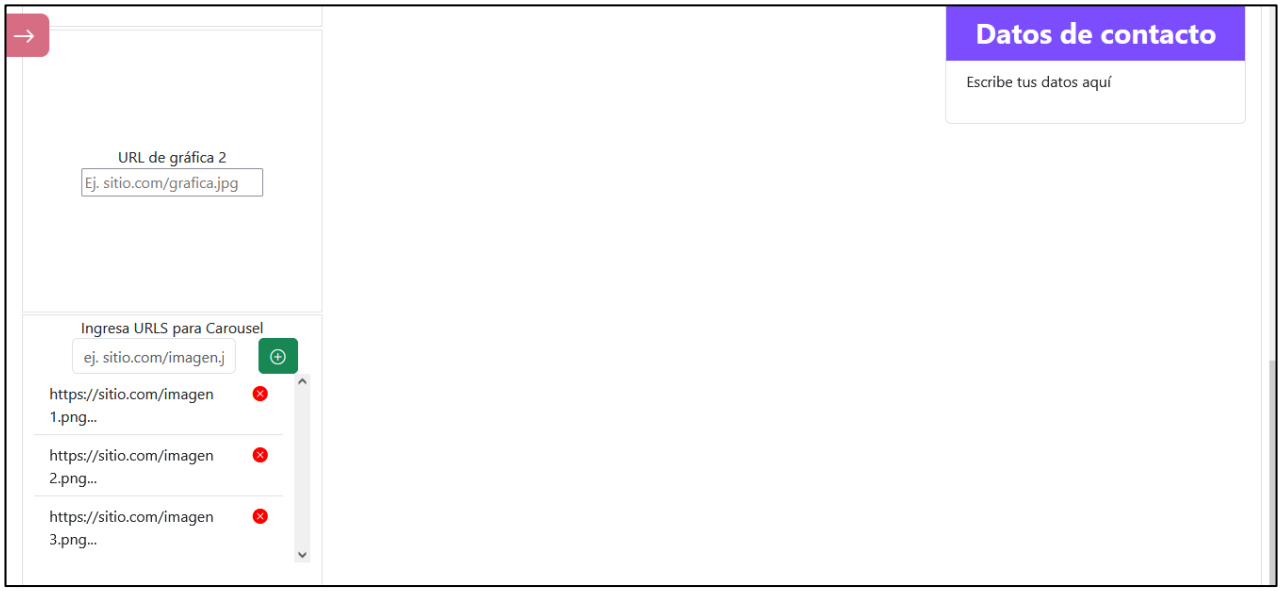



Imagen 28. Segunda parte del formato 2 editable.


Imagen 29. Primera parte del formato 3 editable.

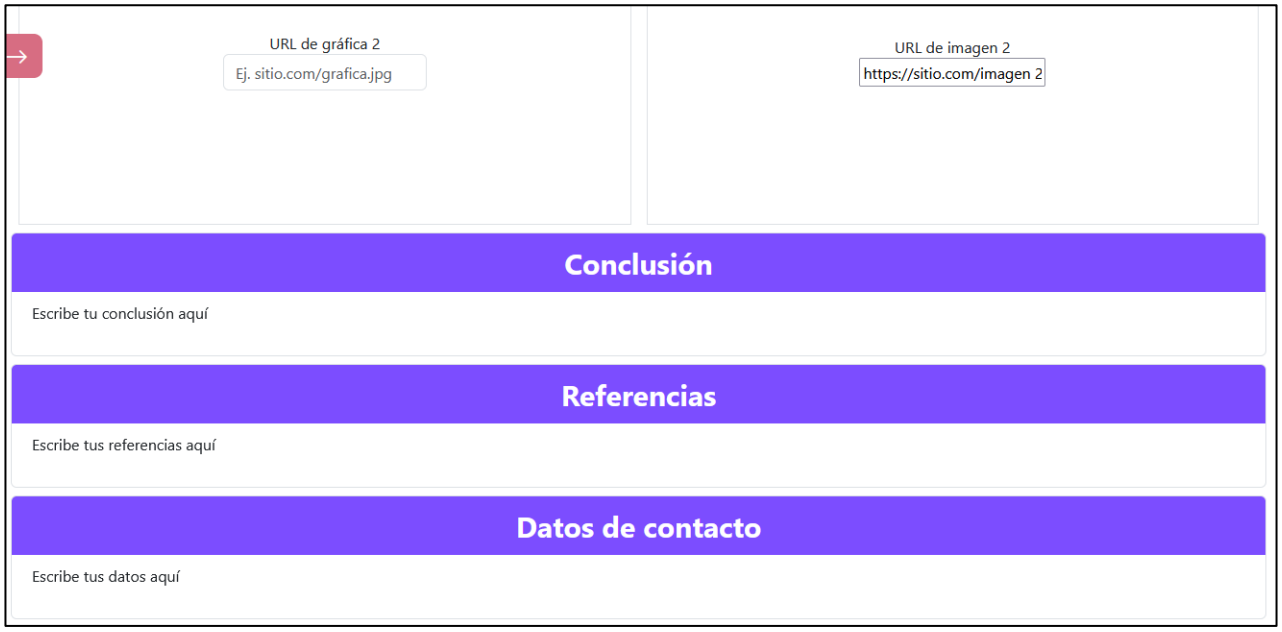



Imagen 30. Segunda parte del formato 3 editable.

Como se puede observar en las imágenes anteriores, la información editada se mantiene gracias a la herencia de los datos entre los formatos y su padre que es *Editable Page Component*, inclusive en el formato 3 tiene una diferencia, en lugar de tener el *Input* Carrusel tiene solo 2 *inputs* que son las dos primeras imágenes que tienen los demás formatos para poder tener integra la información en cada uno de ellos.

Después de haber enlazado la herencia y la opción del *Sidebar* para el cambio entre formatos se realizó lo mismo para el color. Se puede apreciar que al cambiar los colores también se cambia de forma íntegra en cada uno de los formatos según su distribución (Imágenes 31, 32, 33 y 34).

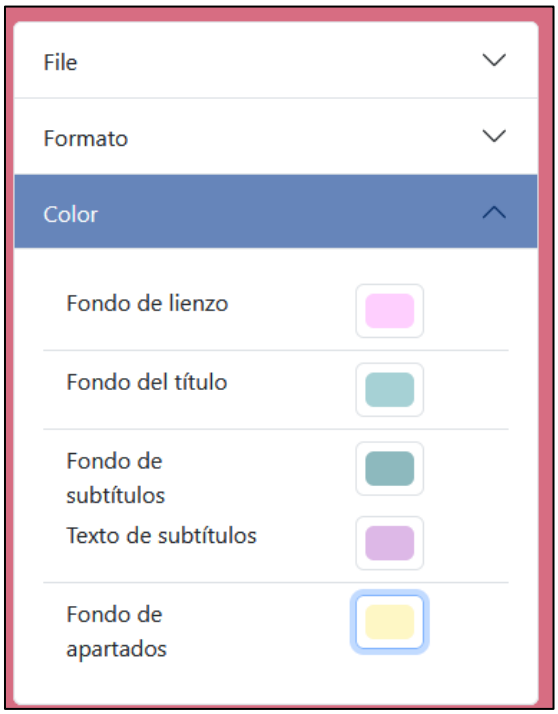



Imagen 31. Cambio de colores en el *Sidebar*.

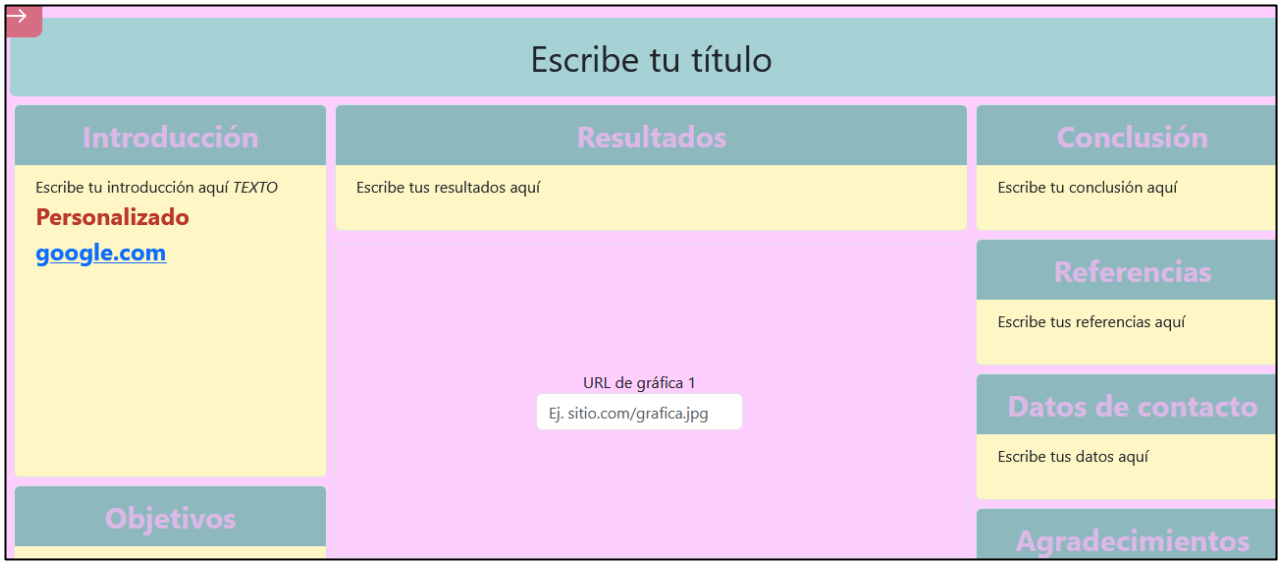



Imagen 32. Cambio de colores en el formato 1.


Imagen 33. Cambio de colores en el formato 2.


Imagen 34. Cambio de colores en el formato 3.

Por último, se creó la función del botón *preview* que permite redireccionar a la página de vista previa que se encuentra en el *Visual Module* cuya ruta de navegación es ***/ver/preview/***.

## Implementación de página de *preview* y cartel en *Visual Module*

Como previamente se describió en la creación de rutas y en la implementación de la página de edición, el *Visual Module* corresponde a una página reutilizable tanto para visualizar el *preview* del *e-cartel* como para mostrar uno previamente guardado en la base de datos ~~en~~ de la página de edición. Una página proviene de la ruta ***/ver/preview*** y la otra de ***/ver/cartel***, esto es muy importante ya que ambas reciben un parámetro diferente para mostrar contenido.

Si en la página *Visual Page Component* se captura la ruta ***/preview*** debe recibir un parámetro cifrado en *base64,* previamente codificado como *URL* y, que al descifrar y decodificar es un objeto de tipo JSON que contiene los enunciados con sus estilos y los colores que se obtuvieron en *Editable Page Component*. Por el contrario, si viene de la

ruta ***/cartel*** recibe un *id* que es un número con el que se insertó la página en la base de datos, al obtener este *id* se hace uso del *Back Service* para obtener dicha información.

El *e-cartel* en general siempre que es recibido o enviado a otro componente se cifra y codifica ya que lo que se está guardando son plantillas de *HTML* combinadas con *CSS* y pueden causar un error al ser recibidas por *URL*. Un ejemplo son los colores que al seleccionarlos se guardan con la nomenclatura hexadecimal de la hoja de estilos e inician con hashtag, esta cadena haría una redirección incorrecta en lugar de enviar los colores a la página de visualización.

Respecto al formato visual para este módulo, se utilizó la creación de componentes en *Angular* de la misma forma que en *Editable Module* ya que ahí está la estructura de los formatos, sin embargo, estos datos ya no son editables, se referencia a la imagen y el Carrusel de imágenes visible se añade a través de la librería de *Bootstrap*. Por otra parte, los márgenes son menores para que se aprecie tal cual como una página completa y no como una página de edición.

Adicionalmente, se crearon dos *Pipes* solo para este módulo, estos ayudan a mostrar la información de una forma condicional de acuerdo con la entrada que se obtiene, dado que no hay una validación por parte del *Editable Module*. Se resolvió a través del *Pipe* de imágenes y el *Pipe* de carrusel que, a grandes rasgos, si el enlace de la imagen resulta ser vacía, envía una imagen por defecto indicando que no se insertó nada como se muestra en las Imágenes 35 y 36.

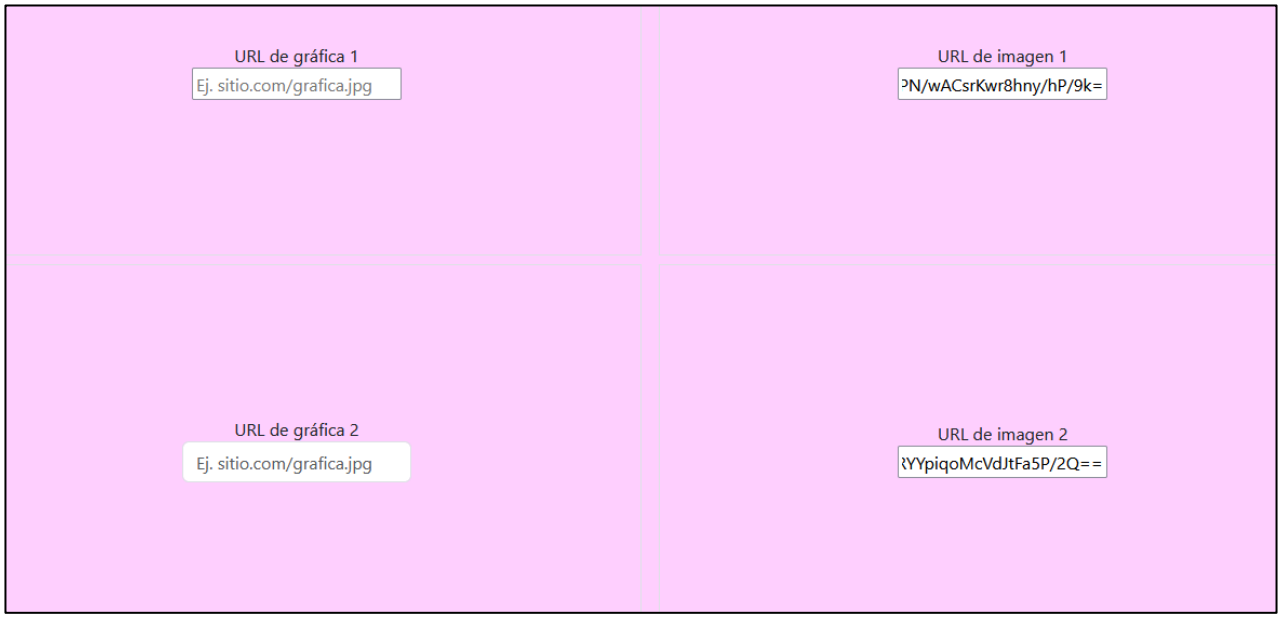



Imagen 35. *Input* de texto para imágenes lleno y gráficas vacías en el formato 3 de *Editable Page Component*.

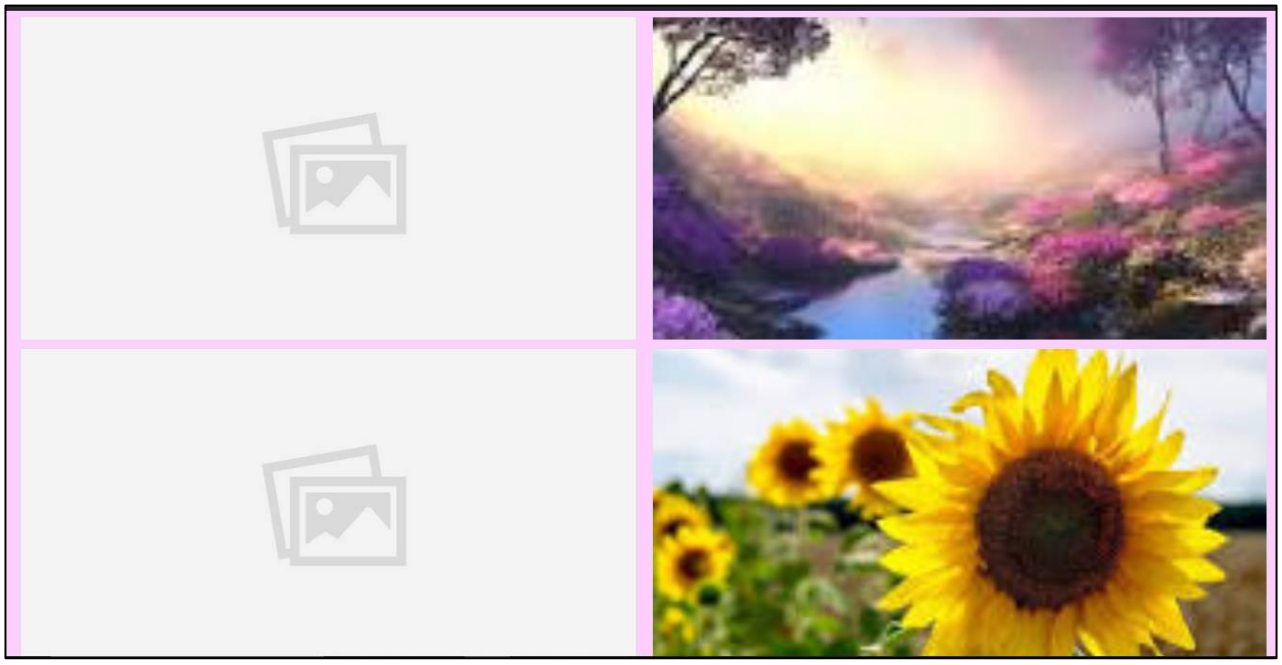


Imagen 36. Resultado del *preview* utilizando *Pipes* que filtran las cadenas vacías poniendo la imagen por defecto.

Como se puede ver a continuación, el *preview* recibe correctamente la información a través del parámetro que contiene codificada y cifrada la información del *e-cartel*, preserva los colores, los datos y los estilos que el usuario desea configurar. Inclusive, no permite editar la información ya que solo es una vista previa para revisar el resultado de su edición y finalmente decida entre guardar, finalizar o continuar editando sin perder la página de edición (Imagen 37).


Imagen 37. *Preview* recibiendo información a través de la *URL* y mostrándola.

Cabe aclarar que para cada uno de los formatos son visibles los cambios en su vista previa. De recibir una cadena no reconocible de JSON para el *preview,* redirecciona a la página 404; para la ruta ***/ver/cartel*** con el *id* solo recibe información del servicio con el *e-cartel* de la base de datos, de lo contrario redirige también a la página de error.

## Implementación de Servicios

Para este proyecto se configuraron dos servicios. El primero es *Design Service,* dedicado a realizar el cambio de formato y el cambio de color. Emite señales de modo que son recibidas de una página hermana del padre a sus hijos, este es el caso de *Layout Page*

*Component* que contiene el *sidebar* y *Editable Page Component* la cual contiene los formatos que son sus componentes hijos. Por lo anterior, se realizaron variables globales que permiten tener un color y formato inicial por defecto para evitar errores de entrada. Las funciones específicas de los servicios de *Design Service* son usadas por el *Layout Page Component* y la suscripción a sus variables de cambio enlazadas a dichas funciones es usada en el *Editable Page Component* y heredados a sus hijos que son los componentes de formato.

El segundo es *Back Service,* dedicado solo a las herramientas de formato del *sidebar*. Este, activa una sola función que funge como señal del *Editable Page Component,* que tiene la información total del *e-cartel*, al pasar al *Layout Page Component* activa el tipo *preview*, guardar o finalizar en la página editable.

El *preview*, obtiene el *e-cartel* por medio de una interfaz en *Typescript*, este contiene la estructura que se manda al *backend*, la información se codifica a *URL,* se cifra en *base64* y por medio de *Router* se envía a una página externa concatenando la ruta de /ver/*preview* con el *e-cartel* ya preparado para que lo reciba la página como anteriormente se explicó.

El guardado, obtiene el cartel en las mismas condiciones que el *preview,* pero, además, recibe su nombre para identificar en qué *id* de la base de datos debe guardarse. Este, a diferencia del *preview* retorna un *Observable* al cual se suscribe el *Edit Page Component* para saber si la respuesta fue exitosa por parte de la *API* o, al contrario, hubo un error (Imagen 38).

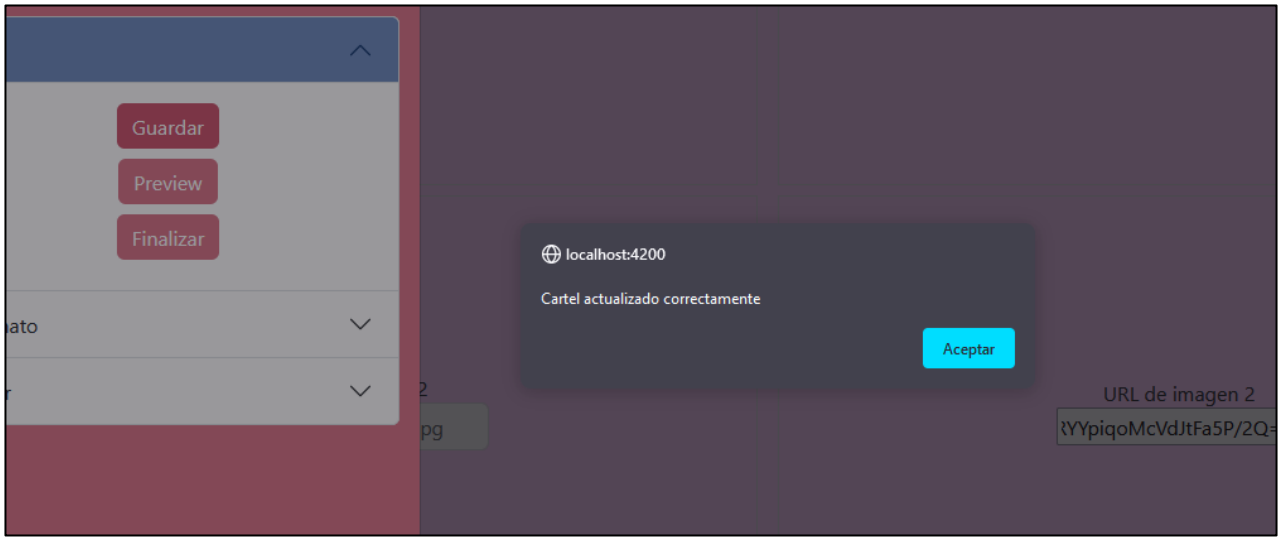



Imagen 38. Respuesta del servicio de guardado tras recibir la señal del botón asociado.

El finalizado, de igual forma que el servicio de guardado, retorna un *Observable* con la respuesta, pero la diferencia es que este, solo recibe el nombre del *e-cartel* para modificar únicamente el campo finalizado a través de la *API*. Este servicio y los anteriores se conectan a los botones del *sidebar* en la sección de *File,* para que haga estas funciones, y el *Edit Page Component* las reciba para enviar la información (Imagen 39).

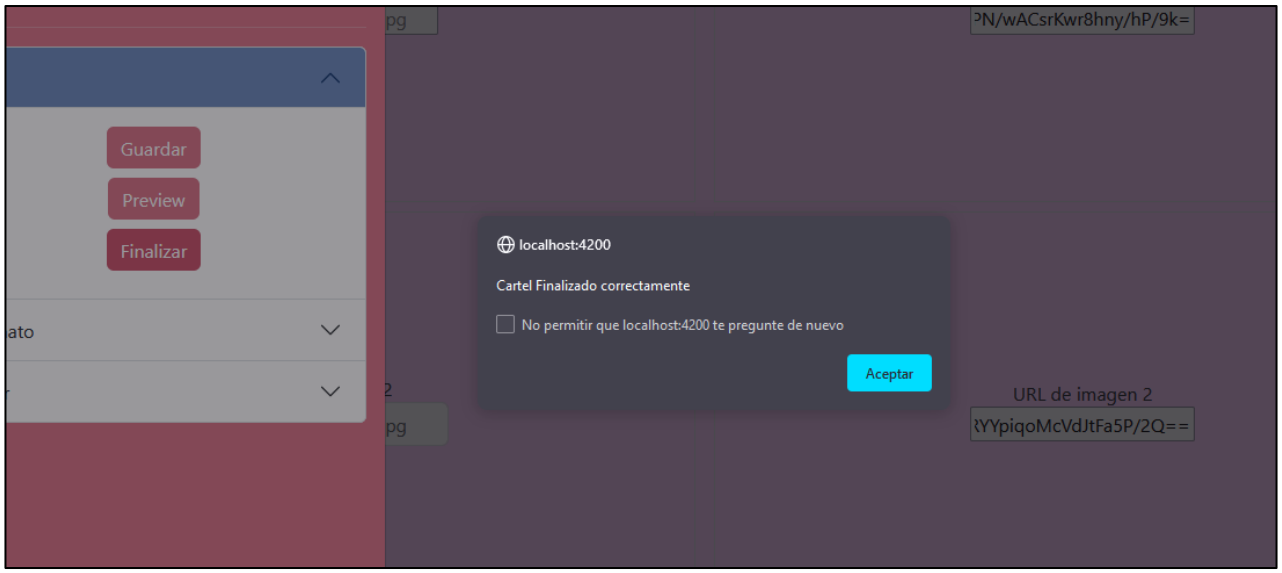



Imagen 39. Respuesta del servicio de finalizado tras recibir la señal del botón asociado.

Además, se añadieron dos servicios, el primero crea el *e-cartel*, es usado por la página principal en el *Home Module*, quien recibe el nombre del nuevo archivo a editar y retorna un *Observable* con una respuesta para hacer saber al usuario si ya existe el nombre de su versión o continuar hacia la página de edición o, por el contrario, mostrar algún otro error por parte de la *API*.

El segundo servicio es para obtener el cartel por medio del *id*, es ocupado por la ruta ***/ver/cartel*** del *Visual Page Component* en su módulo y solamente referencia a la misma página del *preview* de un *e-cartel* guardado, editado previamente y que es consultado desde la base de datos. Este servicio recibe un *id* del *e-cartel* que se quiere consultar, luego retorna un *Observable* con un arreglo de opciones, y al hacer una petición el array devuelve solo uno o varios dependiendo los parámetros que recibe la *API*. El resultado de la petición por medio del servicio es recibido en *Visual Page Component* y plasmado dependiendo del *id,* como se puede ver a continuación (Imágenes 40 y 41).

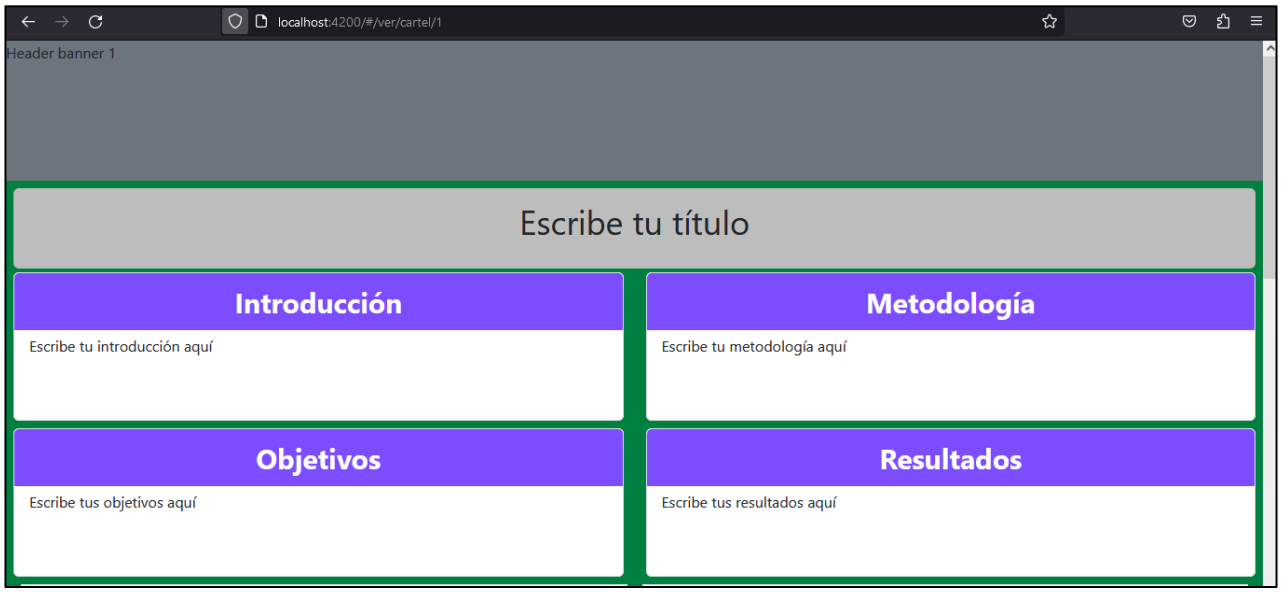



Imagen 40. Visualización del cartel con el *id* igual a 1 previamente guardado en la base de datos.


Imagen 41. Visualización del cartel con el *id* igual a 12 correspondiente al último *id* guardado usado en la implementación del *Design Module*.

## Implementación de diseño responsivo para dispositivos móviles

Como último proceso para complementar el diseño del *frontend,* se ocuparon las clases responsivas que permiten usar *Bootstrap*, dando una columna de 12 secciones para cada elemento para al usuario no se le dificulte editar o visualizar. Se disminuyó el tamaño de letra en *TinyMce* donde el máximo permitido es 38px y reducido a 24px para dispositivos móviles, tanto en edición como en visualización (Imágenes 42, 43 y 44).

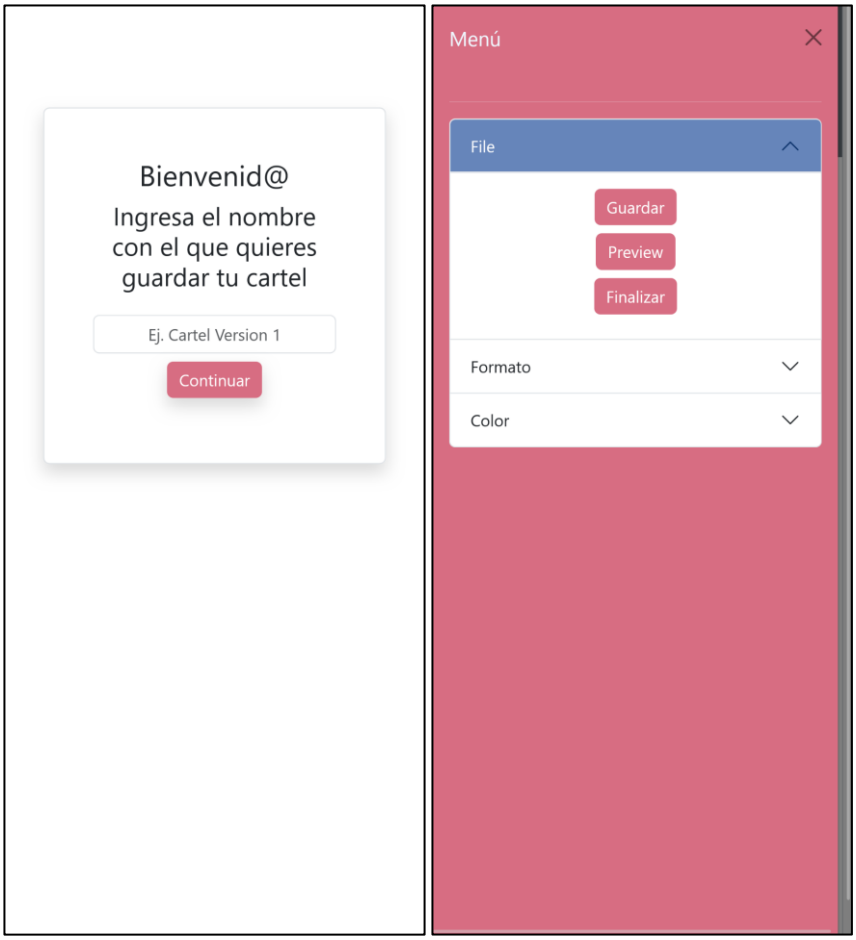



Imagen 42. Vista responsiva para dispositivos móviles de la página principal y s*idebar* de la página de edición.


Imagen 43. Vista responsiva para dispositivos móviles de la página de edición de los 3 formatos diferentes.


Imagen 44. Vista responsiva para dispositivos móviles de la página de visualización de los 3 formatos diferentes.

## Creación de *backend*: Base de datos y *API*

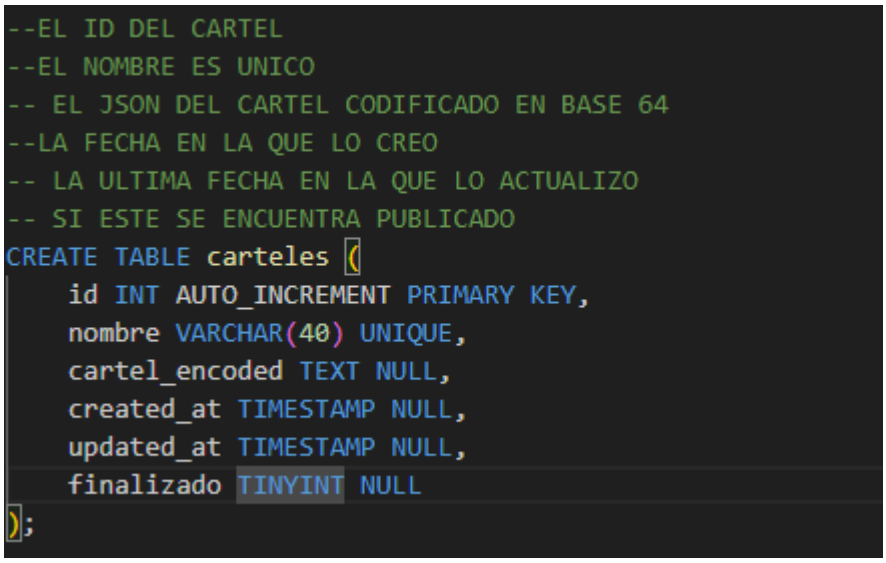
### Creación de Base de datos

Inicialmente, después de haber instalado *AppServ* correctamente se ingresó a *phpMyAdmin* y se creó la base de datos llamada cartel. En el *backend* se usó *AppServ,* que integra *MySQL* y, a través de *phpMyAdmin* se permite la gestión de la base.

### Creación de tabla carteles y *queries*

Posterior a la base de datos, se creó un *query* para generar la tabla llamada ***carteles***. Esta permitirá insertar cada *e-cartel* con los respectivos servicios y, por medio de la *API,* hacer una conexión hacia la base de datos para administrarla. El *query* usado para todas las consultas está alojado en la aplicación de *Angular,* en la ruta /back/create_tables.sql, aquí se encuentran las instrucciones para crear la tabla: insertar, actualizar y eliminar ~~dentro de ella~~, así como truncar la misma.

Como se aprecia en la Imagen 45, para que el servicio del nuevo *e-cartel* sea exitoso debe permitir que los demás campos, a excepción del nombre, sean opcionales de llenar, así, al momento de guardar serán actualizados sin problema. Además, el *e-cartel* se integra como un tipo *TEXT,* ya que este tipo de dato no tiene limitación al cifrar y codificar. A diferencia del *varchar* que solo permite hasta 40 caracteres en el nombre. Para el *Visual Page Component* el *id* es un dato imprescindible para obtener el *e-cartel* y visualizarlo correctamente.



```
--EL ID DEL CARTEL
--EL NOMBRE ES UNICO
-- EL JSON DEL CARTEL CODIFICADO EN BASE 64
--LA FECHA EN LA QUE LO CREO
-- LA ULTIMA FECHA EN LA QUE LO ACTUALIZO
-- SI ESTE SE ENCUENTRA PUBLICADO
CREATE TABLE carteles (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(40) UNIQUE,
    cartel_encoded TEXT NULL,
    created_at TIMESTAMP NULL,
    updated_at TIMESTAMP NULL,
    finalizado TINYINT NULL
);
```

Imagen 45. Instrucción para crear la tabla dentro de la base de datos.

## Implementación de *API* y configuración de conexión a la base de datos

Inicialmente se programó un archivo de configuración en *PHP* que hace la conexión entre la base de datos, este se importa a cada *endpoint* cuando se hace una petición a estos. Como primera función se usaron *headers* para que no permita el acceso desde la aplicación en *localhost* por medio del puerto 4200, el cual despliega *Angular* para su desarrollo, dichos puertos pueden cambiar y variar en producción.

Después, se configuró la variable de conexión a la base de datos, a través de la función *MySQL* con los datos del puerto, el usuario y *password*. Además, se creó una función que permite mostrar los resultados de un arreglo para la respuesta que regresará a través de la función ***echo*** (Imagen 46).

```php
<?php
    header("Access-Control-Allow-Headers: Content-Type");
    header("Access-Control-Allow-Origin: http://localhost:4200"); //AQUI TAMBIEN SE CAMBIA
    header("Access-Control-Allow-Methods: OPTIONS,POST,GET");

    $host_name = 'localhost';
    $database = 'cartel';
    $user_name = 'root';
    $password = 'password';
    $port='3306'; //Podría ser otro

    /*
        * Escribiremos un código controlado, que vaya evaluando las variables
        * Nunca podemos dar por hecho que las cosas funcionarán porque sí
    */

    /* OBJETO CONEXIÓN */
    $connect = new mysqli($host_name, $user_name, $password, $database, $port);
```

Imagen 46. Archivo de configuración para conexión a la base de datos local en *AppServ*.

## Implementación de *endpoints* para la *API*

Para el proyecto existen cuatro *endpoints* en la *API* y parten de la carpeta *back,* mencionada previamente, para conectar los servicios y consumir dicho *endpoint*. Los *endpoints* principales vienen de su ruta absoluta del servidor de *AppServ* donde están alojados y son: *nuevo_cartel*, *update_cartel*, *get_cartel* y *finalizar_cartel*. Para cada uno se usaron diferentes consultas en *MySQL:* insertar, actualizar y consultar, principalmente al preparar el *query* a ejecutar y, posteriormente, al capturar la respuesta que arroja la base de datos de tal forma que el *frontend* comprenda la respuesta y la interprete como se explicó en la implementación de *Angular*.

Para el *endpoint nuevo_cartel,* primero en usarse, se obtienen los parámetros por medio del *body* que es únicamente el nombre; si esta asignación ya existe, no lo inserta y manda un mensaje de error, de lo contrario, regresa el mensaje de que se ha creado correctamente y es recibida en el *Back Service* para continuar a la página de edición. Finalmente, la fecha de creación es construida en este archivo *PHP* e insertada en la base de datos (Imágenes 47 y 48).

```
//CREAR NUEVO CARTEL RECIBE SOLO EL NOMBRE
newCartel(name:string):Observable<Res>{
  //ENVIA EL NOMBRE AL SERVICIO
  return this.http.post<Res>(`${g.API_CORE}nuevo_cartel.php`,{ name: name });
}
```

Imagen 47. Servicio de crear cartel haciendo petición al *endpoint* de nuevo_cartel.

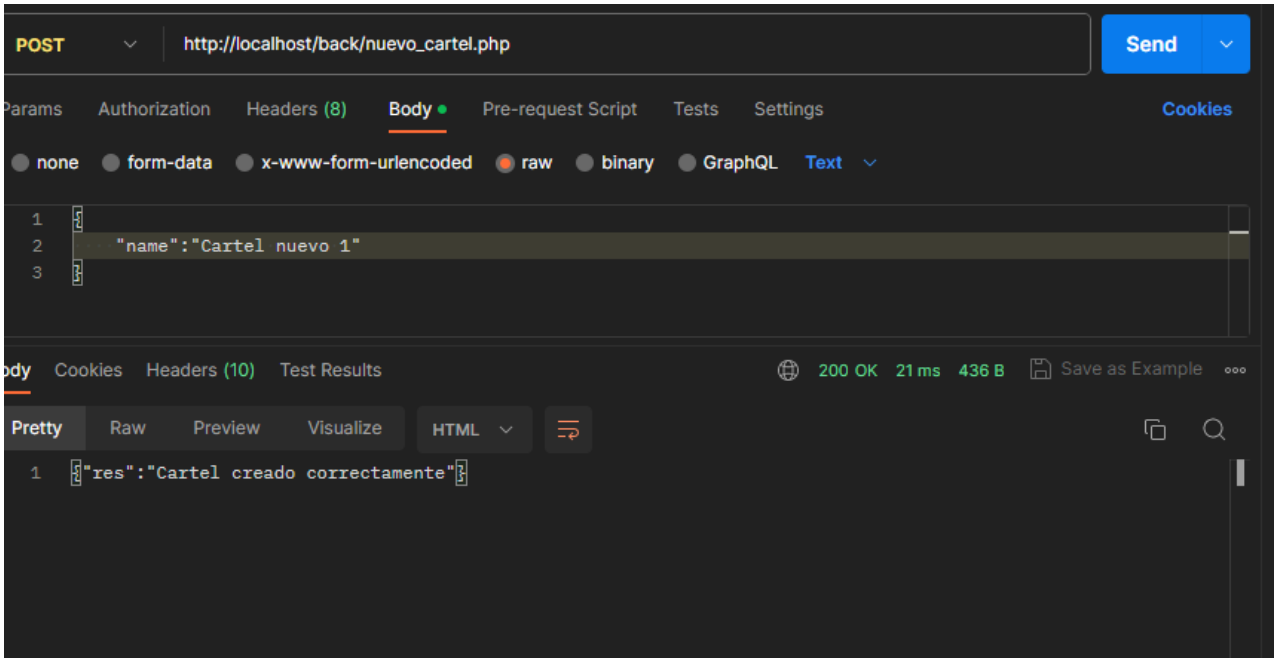



Imagen 48. Ejemplo de petición post al *endpoint* nuevo_cartel.

El *endpoint* update_cartel, usado al dar *clic* en guardar, de igual forma obtiene los parámetros por medio del *body* que son el nombre y el cartel; de existir el nombre, lo actualiza y manda un mensaje de guardado, de lo contrario, regresa el mensaje de que no existe dicho archivo. Este control de flujo es controlado por el *frontend* ya que no se permite llegar a la página sin obtener un nombre anteriormente. Respecto a la fecha de última actualización con la que se llena el registro del *e-cartel* a insertar es construida en este archivo *PHP* e insertada en la base de datos (Imágenes 49 y 50).

```
//GUARDAR CARTEL (ACTUALIZAR) RECIBE EL NOMBRE Y EL CARTEL
saveCartel(name:string,cartel:Cartel):Observable<Res>{
  let cartel_b64=btoa(encodeURIComponent(JSON.stringify(cartel)));

  return this.http.post<Res>(`${g.API_CORE}update_cartel.php`,{
    name: name,
    cartel:cartel_b64
  });
}
```

Imagen 49. Servicio de guardar de cartel haciendo petición al *endpoint* de update_cartel.

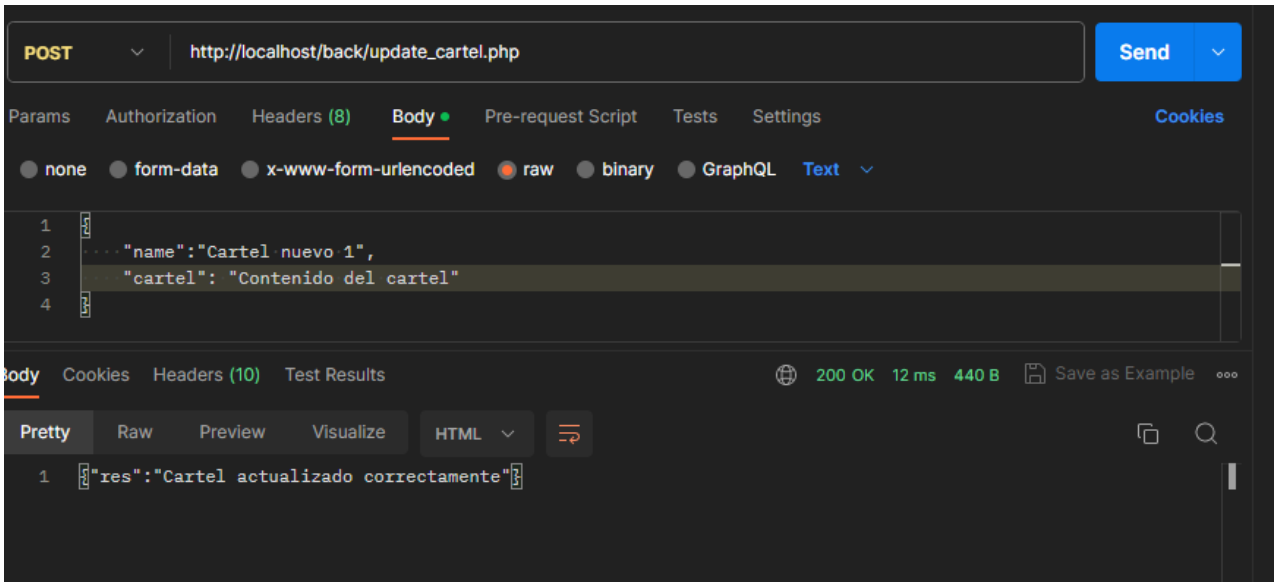



Imagen 50. Ejemplo de petición post al *endpoint* update_cartel.

En cuanto al *endpoint* finalizar_cartel, usado al dar *clic* en finalizar, de igual forma obtiene los parámetros por medio del *body* que es solamente el nombre y, de existir este, lo actualiza y manda un mensaje de finalizado, de lo contrario, regresa el mensaje de que no existe dicho archivo, el control de flujo es controlado al igual que el de guardado (Imágenes 51 y 52).

```
//PONER EN FINALIZADO SOLO RECIBE EL NOMBRE
finalizeCartel(name:string):Observable<Res>{
  return this.http.post<Res>(`${g.API_CORE}finalizar_cartel.php`,{
    name: name
  });
}
```

Imagen 51. Servicio de finalizar cartel haciendo petición al *endpoint* de finalizar_cartel

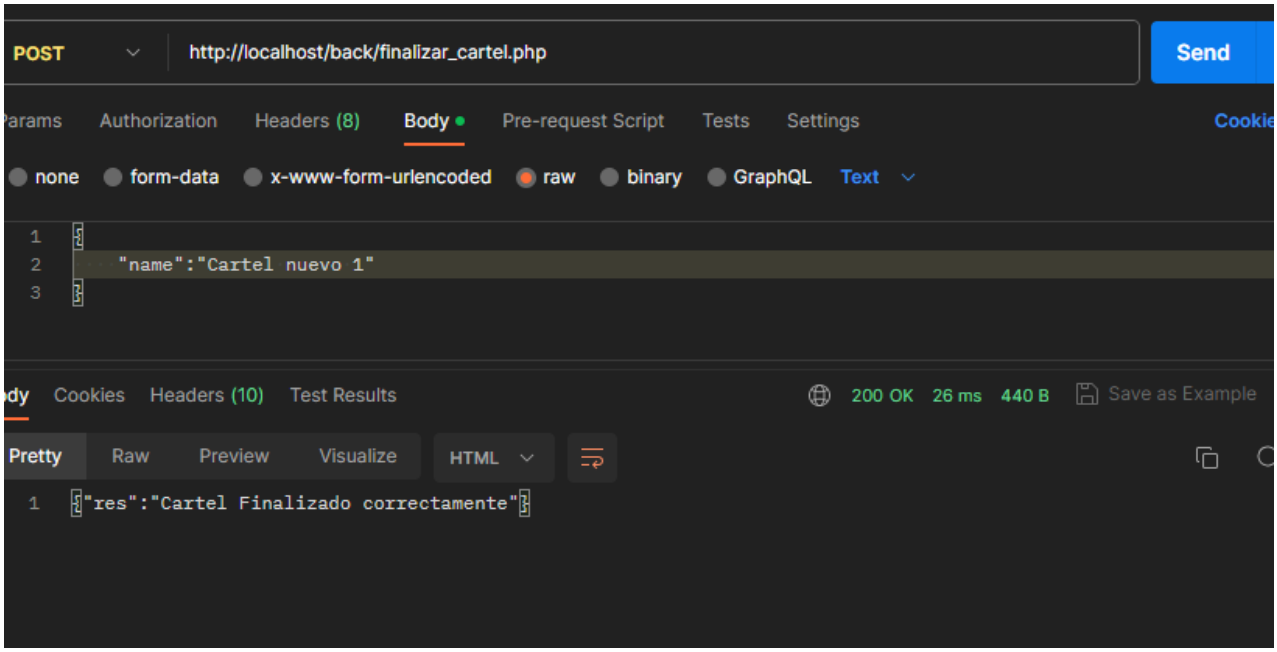



Imagen 52. Ejemplo de petición post al *endpoint* finalizar_cartel.

Por último, el *endpoint* de get_cartel recibe el *id* del *e-cartel* que se quiere consultar y el campo *byId*, este último es *true* o *false*, devuelve solo un *e-cartel* por *id* que se envió, ~~y~~ de lo contrario, si es *false* envía un array de todos los *e-carteles* existentes. Este es usado en *Visual Component* pero con el *byId* en *true* para que devuelva el consultado por *URL* en dicha página (Imágenes 53 y 54).

```
//OBTENER CARTEL POR ID RECIBE SOLO UN NUMERO Y SE OBTIENE UNA RESPUESTA O UN ARRAY DE CARTELES
getCartel(id:string,byId:string):Observable<any>{
  // console.log(
  //     {id: id,
  //   byId: byId}
  // );
  let body={
    id: id,
    byId: byId
  };

  return this.http.post<any>(`${g.API_CORE}get_cartel.php`,
  body);
}
```

Imagen 53. Servicio de obtener cartel haciendo petición al *endpoint* de get_cartel.

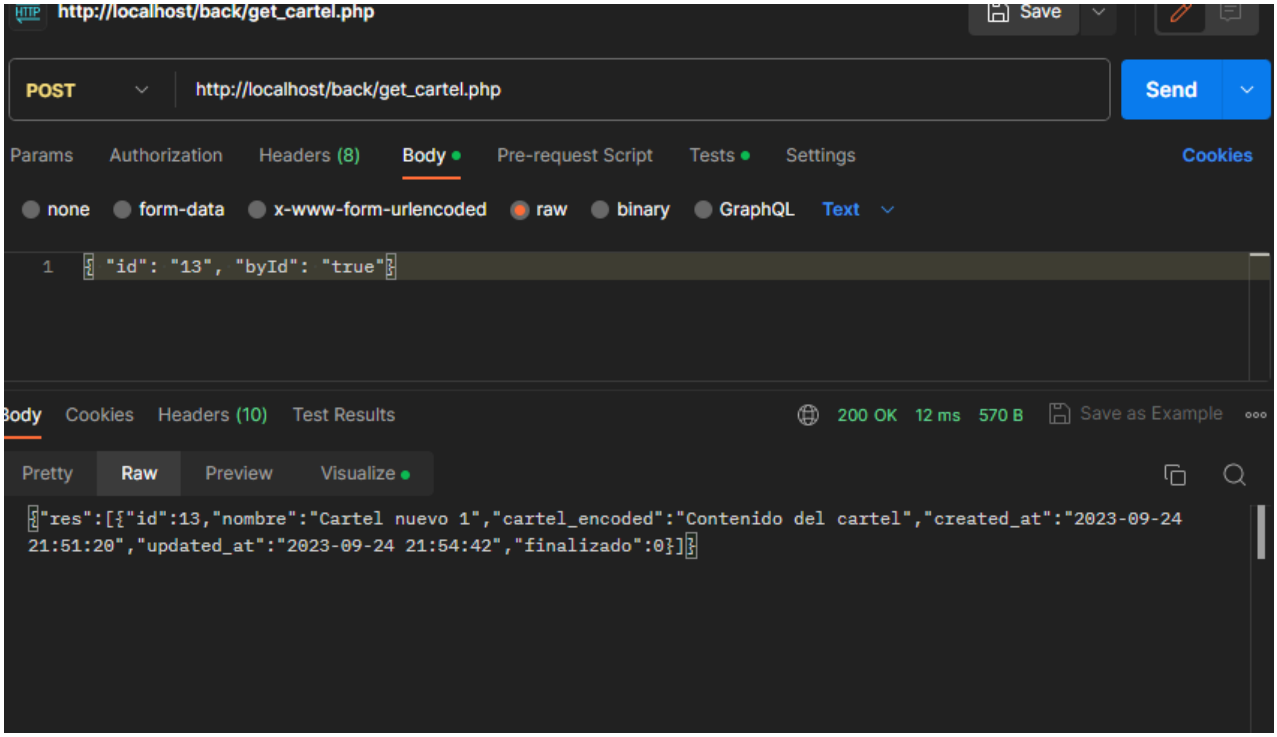



Imagen 54. Ejemplo de petición post al *endpoint* get_cartel.

## Pruebas de funcionamiento

Como último proceso, se realizaron las pruebas con la aplicación de la versión para productivo en *Angular* y así, desplegarla en un servidor local de *AppServ* con las configuraciones respectivas para su buen funcionamiento. Este proyecto contiene un

archivo de *readme.md* con la documentación básica de cómo ejecutar el *frontend* y el *backend,* y qué archivos se modifican para apuntar al *endpoint* final al cual se realizan las peticiones para los servicios programados en *Angular*.

## Despliegue en servidor local *AppServ*

Para poder desplegar la carpeta productiva, generada por *Angular* en un servidor, es necesario que primero se ejecute la directiva ***ng build*** que permite compilar en archivos más compactos las librerías y módulos usados durante el desarrollo, además de integrar la carpeta que contiene archivos externos como las imágenes en *assets*; incluso transcribe el código a un solo archivo *Javascript*, como también lo hace con la hoja de estilos al realizar un proceso llamado *minify uglify* del código, en otras palabras, se reduce el tamaño de código y aumenta el rendimiento y transcribe los archivos a un formato con poca legibilidad de las funciones, haciendo que sea más seguro si se intenta hacer un ataque de inyección de código en *Javascript* (Imagen 56).

```
C:\Users\josue.ramirez\Desktop\Curso\cartel\Cartel>ng build
√ Browser application bundle generation complete.
√ Copying assets complete.
√ Index html generation complete.

Initial Chunk Files           | Names                |  Raw Size | Estimated Transfer Size
main.3c75c6fbdf1232aa.js      | main                 | 291.72 kB |                77.89 kB
polyfills.7e4d9180cc6fd6f7.js | polyfills            |  33.01 kB |                10.62 kB
runtime.211221360a2a6e9e.js   | runtime              |   2.72 kB |                 1.31 kB
styles.212980c3bb2fd0b3.css   | styles               | 726 bytes |               293 bytes

                              | Initial Total        | 328.16 kB |                90.11 kB

Lazy Chunk Files              | Names                |  Raw Size | Estimated Transfer Size
618.a22a2af33b8275ad.js       | design-design-module |  27.92 kB |                 4.44 kB
958.9af35dd14a49df46.js       | visual-visual-module |  18.04 kB |                 3.06 kB
common.5101188e32f4dd35.js    | common               |   5.12 kB |                 1.50 kB
640.6d5d7af3c5ee388c.js       | home-home-module     |   4.08 kB |                 1.52 kB
```

Imagen 56. Compilación de la aplicación para productivo en *Angular*.

En la Imagen 57, se ilustra cómo en el proyecto en *Angular* para productivo, que está dentro de la carpeta *dist,* se encuentra una carpeta del compilado con el nombre del proyecto. Esta contiene los archivos que deben subirse a productivo con el proceso previamente comentado para una mejor optimización de la página.

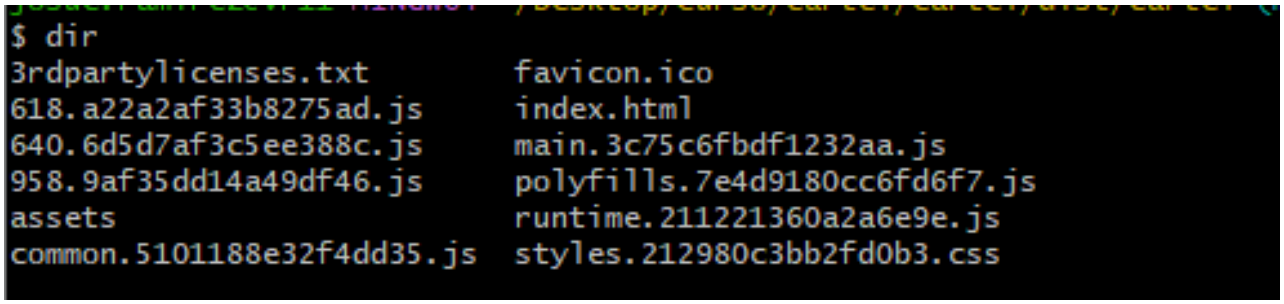



Imagen 57. Directorio de archivos dentro del folder /dist/cartel.

Ahora que se tiene el folder listo para desplegarse en el servidor local, solo es necesario arrastrar la carpeta a ***/AppServ/www***. Dentro de dicha carpeta se encuentra ***/back,*** esta contiene los *endpoints* que conectan con la base de datos. Además, ya se encuentra activa la base de datos **cartel** y se ha creado la tabla **carteles** que fue usada previamente (Imagen 58).

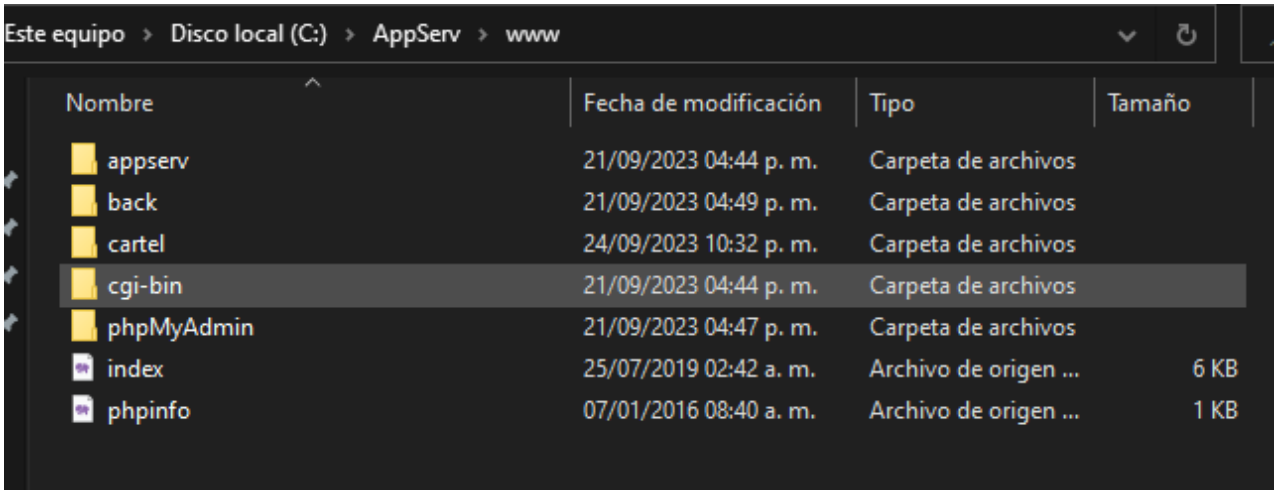



Imagen 58. Carpeta para despliegue de la página en servidor local.

## Página principal y redirección a rutas sin acceso

Después de haber desplegado el compilado para productivo de *Angular* en el servidor, así como la carpeta que contiene los *endpoints* y conexión a la base de datos para la *API*, ya se puede observar que ***http://localhost/cartel*** redirige a ***http://localhost/cartel/#/home/new-page*** correctamente. Además, si se intenta entrar a la página de edición sin ingresar un nombre, redirecciona nuevamente al formulario para

que este se ingrese y, que al ingresar a otra ruta envíe al usuario a la página de 404. Por otro lado, las validaciones por parte del *frontend* y en respuesta de la *API* para obtener e insertar el nombre de un nuevo cartel son correctas (Imagen 59).


Imagen 59. Visualización de la página de error 404 desplegada en el servidor local después de intentar ingresar a http://localhost/cartel/#/nkjdaksb.

Como se puede observar en la Imagen 60, una vez que se ha desplegado la aplicación productiva, el puerto 4200 que pertenecía a *Angular* ya no aparece, sino que al ser parte del servidor de origen no es necesario redirigirla y, por otra parte, el hashtag que parte de la raíz del proyecto ***/cartel/#*** permite redireccionar correctamente a lo configurado durante la implementación del *Routing Module* y *Lazy Loadi*ng de la aplicación.


Imagen 60. Visualización de la página principal desplegada en el servidor local.

## Página de edición para cartel

La página de edición funciona correctamente, así como el uso de estilos para la personalización de textos y colores en cada formato, como se puede observar la Imagen 61, cada herramienta realiza su función de forma reactiva mostrando en ese momento las variaciones hechas con las herramientas del *Sidebar*. En consecuencia, el diseño responsivo se mantiene como se propuso en la implementación durante el desarrollo previo a la producción. También, funciona de forma correcta el uso de los *inputs* para imágenes estáticas y para carrusel de imágenes en el ambiente del servidor local.

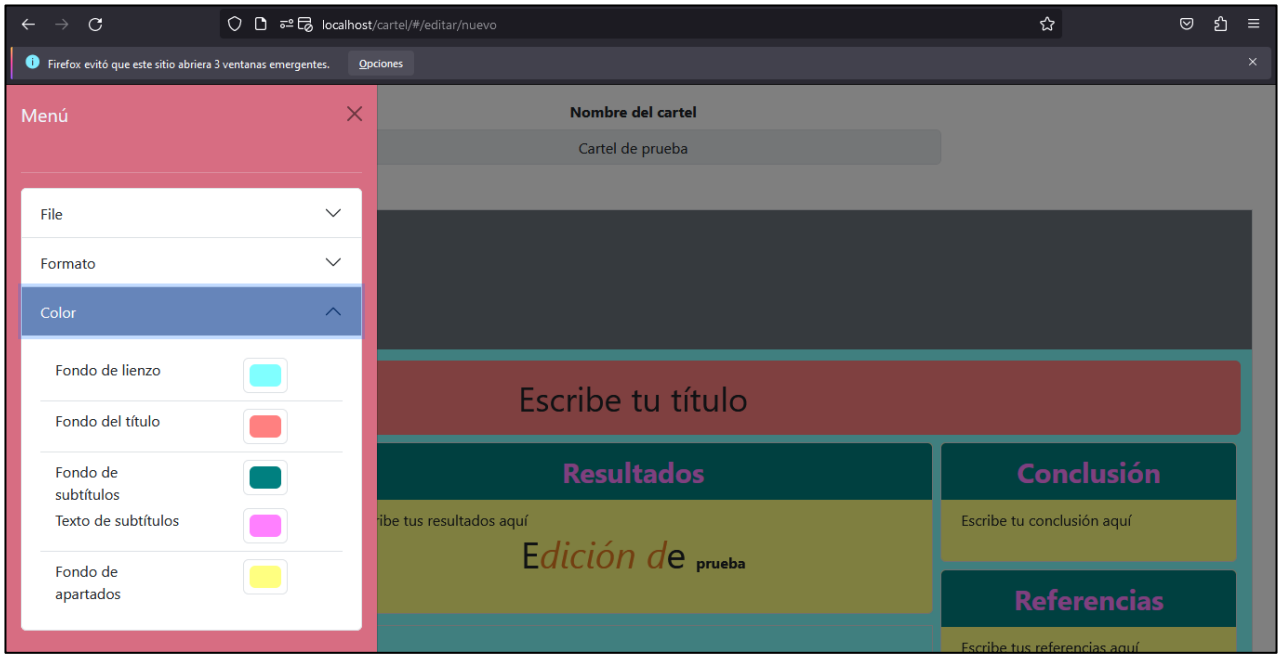



Imagen 61. Visualización y funcionamiento de la página de edición con *Sidebar* mostrada desplegada en el servidor local.

## Guardado y actualización de cartel

Así mismo, se puede corroborar que el servicio de guardado funciona eficientemente y se puede visualizar cómo se inserta dicho registro en la Imagen 62 dentro de la base de datos, después de presionar el botón respectivo del *Sidebar*. En donde la Imagen 63 estaba en 0 y pasa a 1 al ser finalizado, significa que el *endpoint* y el servicio que lo consume están funcionando como se desea. Por otra parte, el servicio de finalizado también concluye con éxito y se ilustra en las Imágenes 64 y 65.

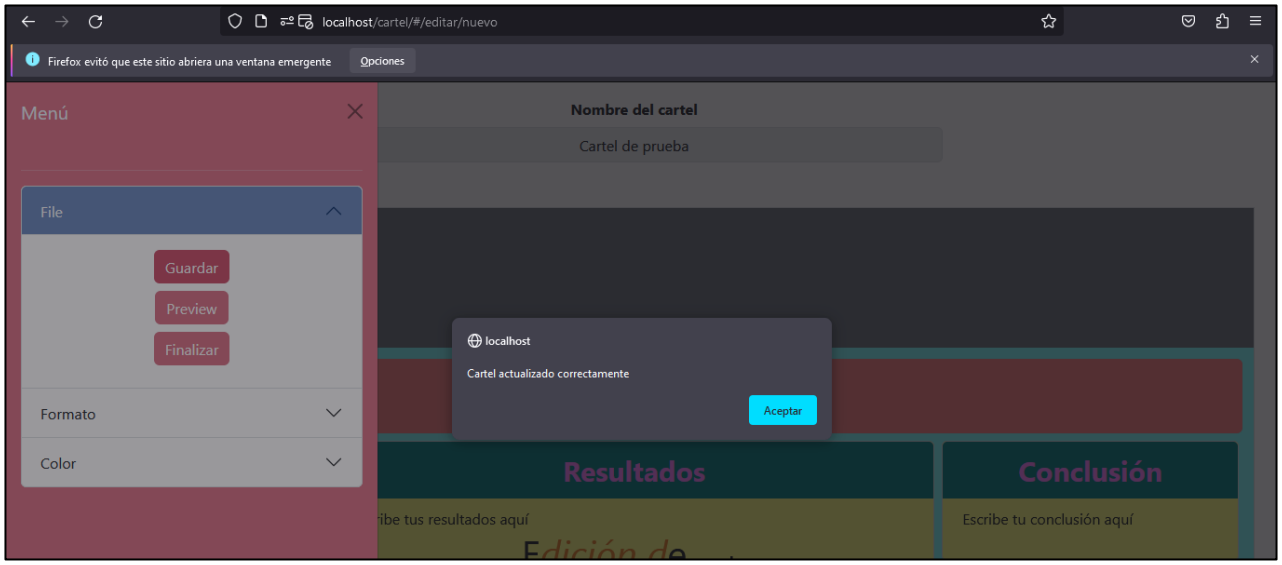



Imagen 62. Visualización y funcionamiento del servicio de guardado en la página de edición una vez desplegada en el servidor local.

| | | | | | |
|---|---|---|---|---|---|
| 14 | Cartel de prueba | JTdCJTlydGl0bGUlMjIlM0ElMjIlM0NwJTlwY2xhc3MlM0QlNU... | 2023-09-25 04:59:09 | 2023-09-25 05:02:15 | 0 |

Imagen 63. Consulta del cartel guardado con los datos recabados del *frontend* en la base de datos.

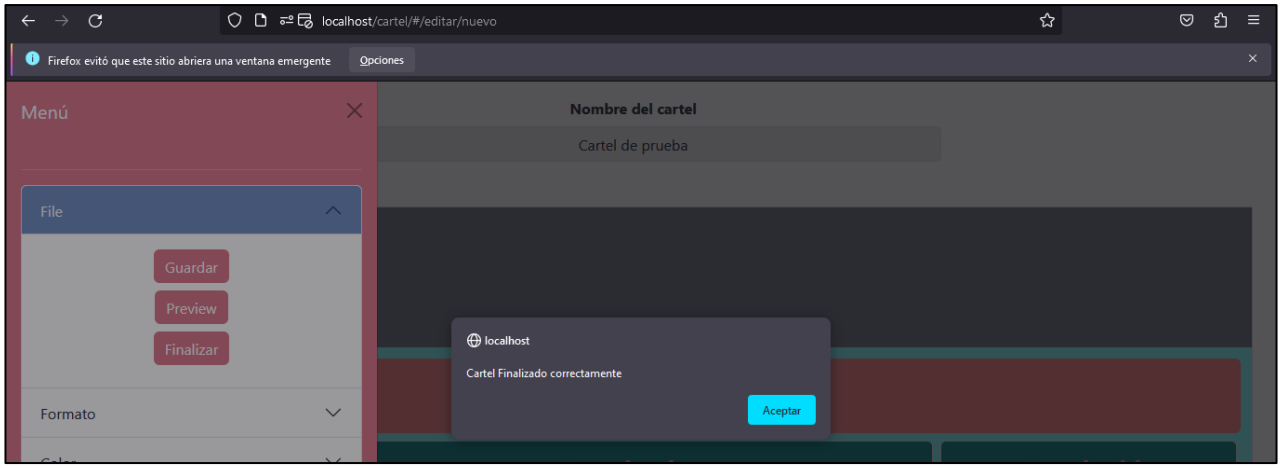



Imagen 64. Visualización y funcionamiento del servicio de finalizado en la página de edición una vez desplegada en el servidor local.

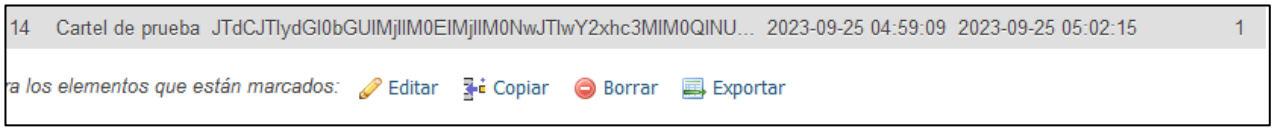



Imagen 65. Consulta del cartel finalizado en la base de datos.

## Vista previa y página final del cartel

En lo que respecta a la vista previa y el cartel final ya guardado se puede observar cómo se despliega correctamente y cómo el *Pipe* que se usó para el control de imágenes también está siendo efectivo cuando no se envía ninguna cadena en particular, haciendo que se muestre una imagen por *default* que hace referencia a que no hay algo que mostrar debido a que el usuario no ingresó ningún enlace de imagen y, de la misma manera, si existen las muestras como previamente se probaron en la implementación (Imágenes 66 y 77).

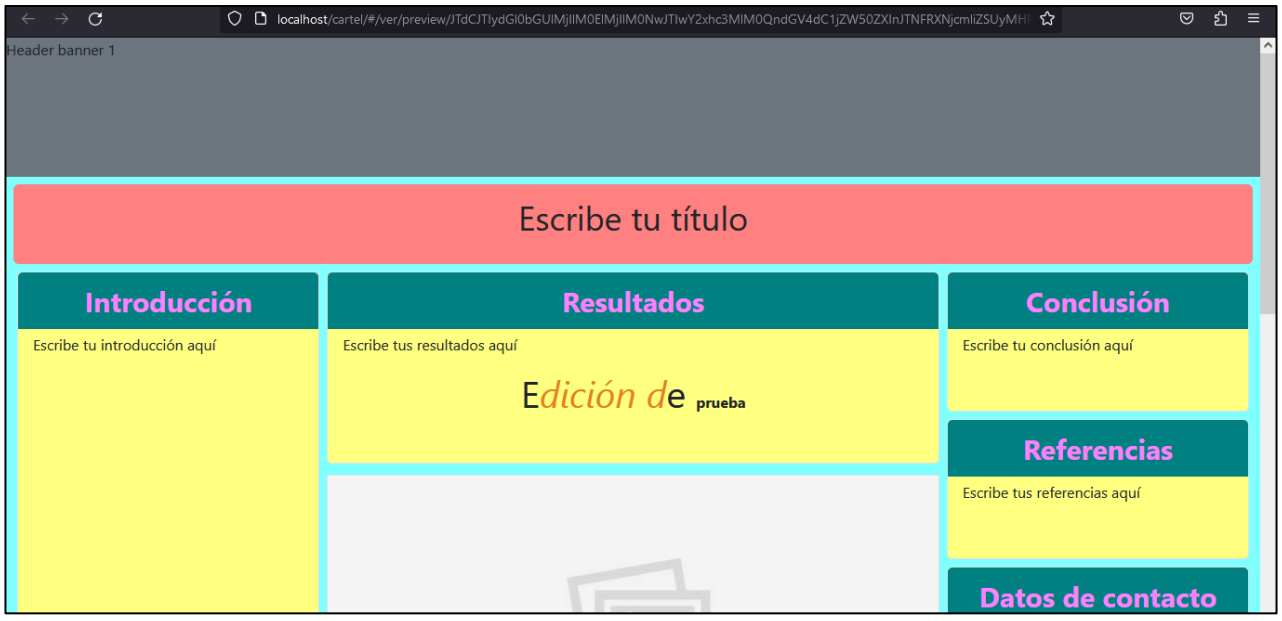



Imagen 66. Visualización en la página de *preview* una vez desplegada en el servidor local.

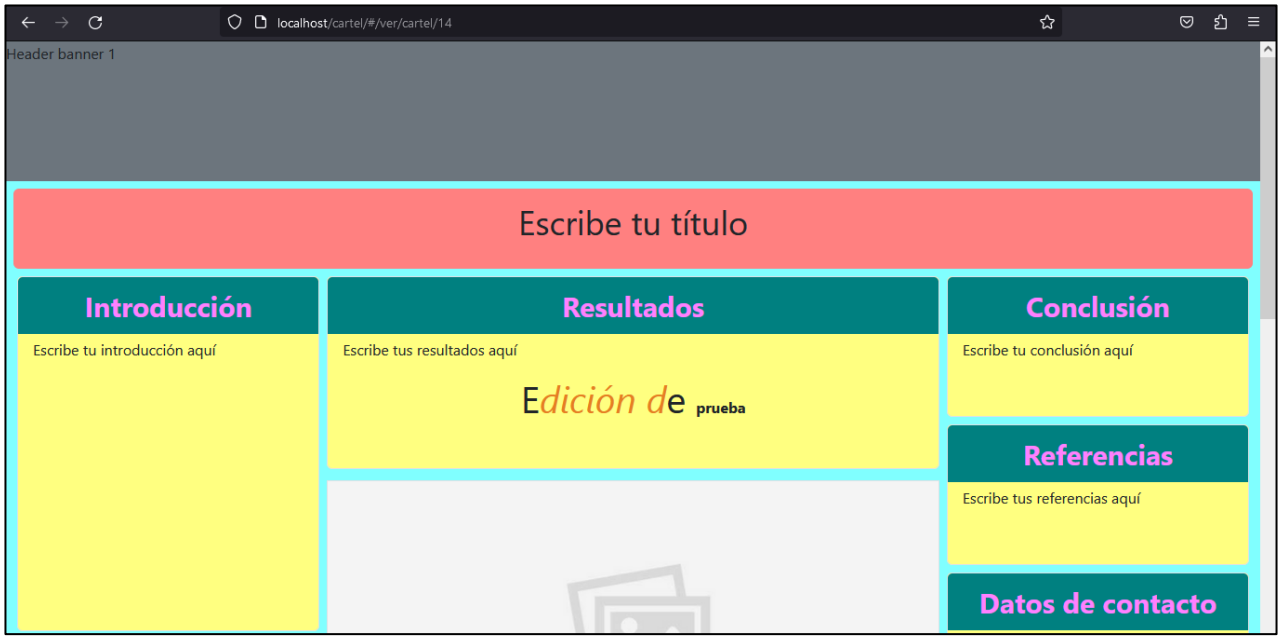



Imagen 67. Visualización en la página de cartel final una vez desplegada en el servidor local.

De igual manera, si se pone contenido visual para el carrusel y para las imágenes estáticas del *e-cartel,* funciona como se desea al controlarlas desde la entrada del cartel, de lo contrario, se muestra la imagen por defecto que el *Pipe image* y el *Pipe image array* usan como solución al filtro de haber añadido un enlace vacío. Los datos ingresados se mantienen conforme al cambio de formato como se puede observar en las siguientes ilustraciones (Imágenes 68 al 73).

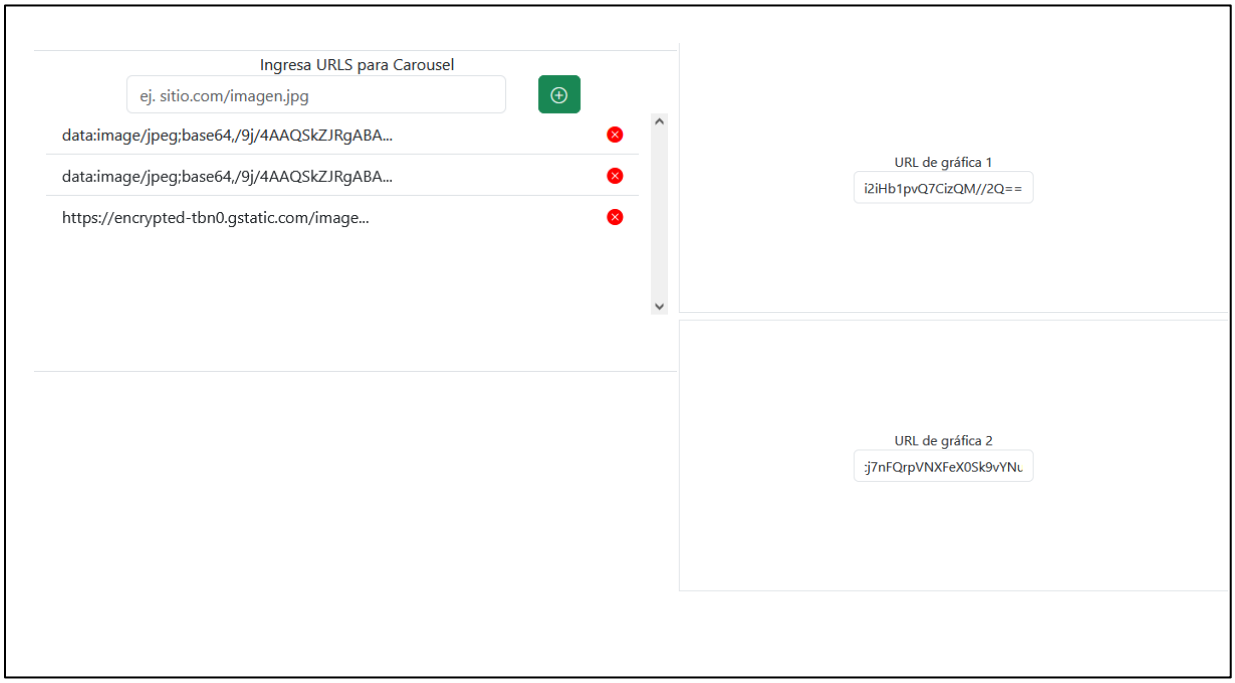



Imagen 68. Entrada para gráficas e imágenes de carrusel en el formato 1.

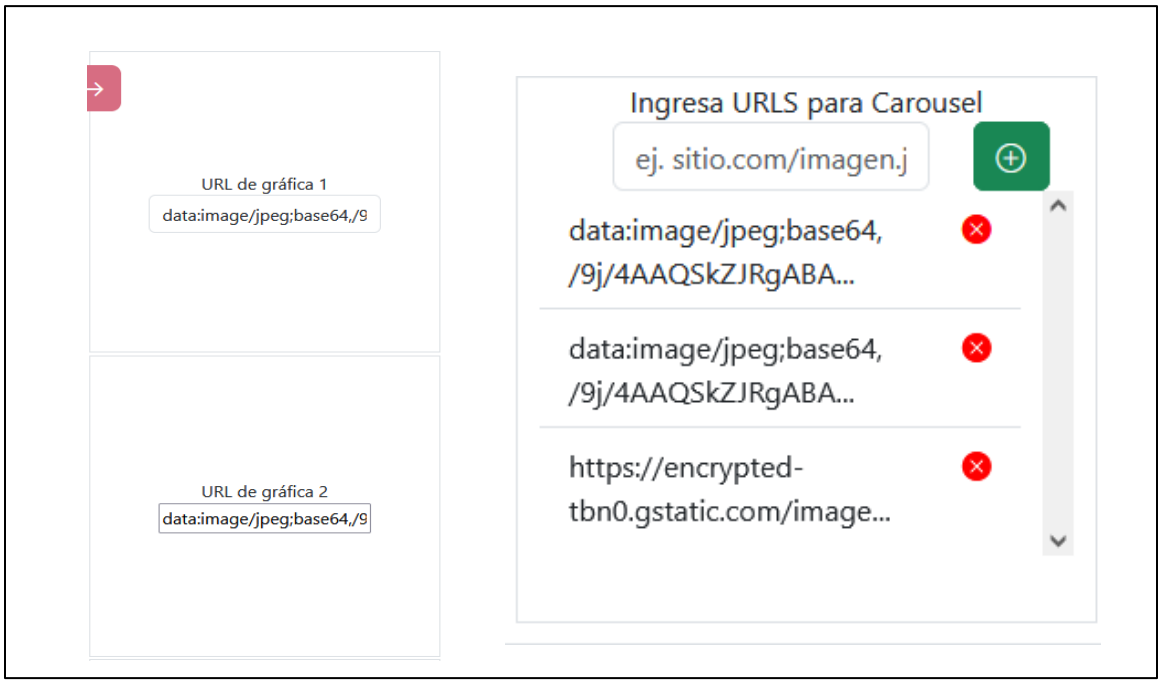


Imagen 69. Entrada para gráficas e imágenes de carrusel en el  formato 2.


Imagen 70. Entrada para gráficas e imágenes en el  formato 3.

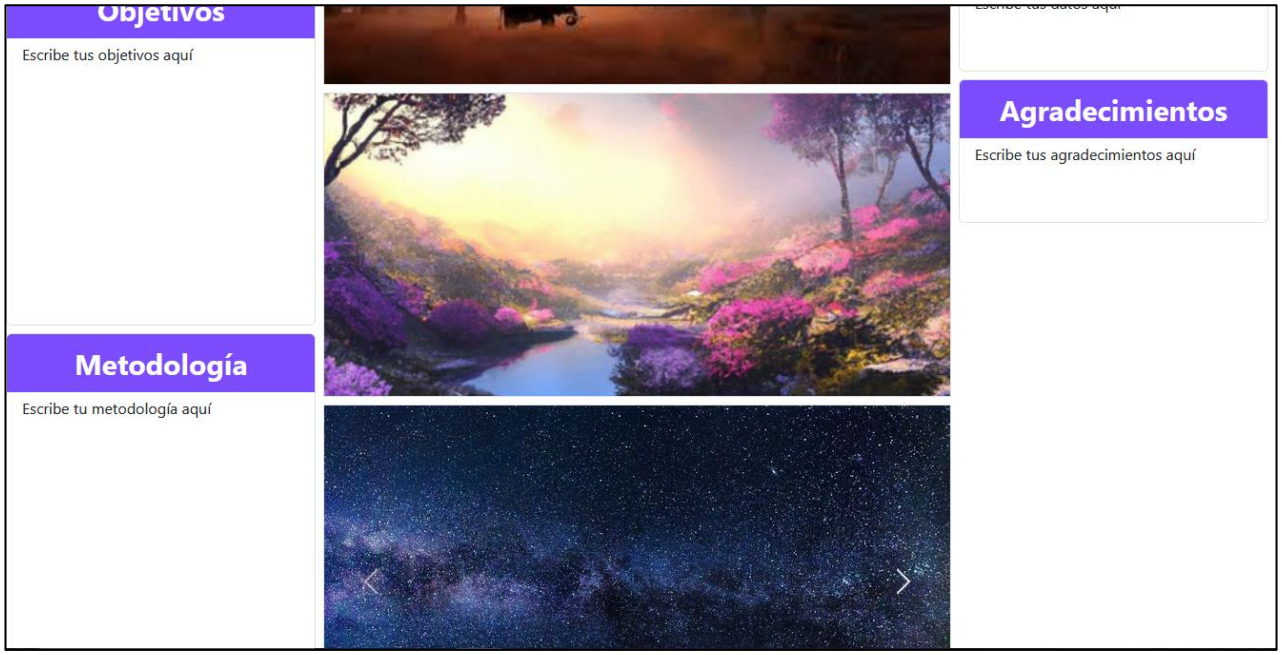


Imagen 71. Resultado de visualización para gráficas e imágenes de carrusel en el formato 1.


Imagen 72. Resultado de visualización para gráficas e imágenes de carrusel en el formato 2.

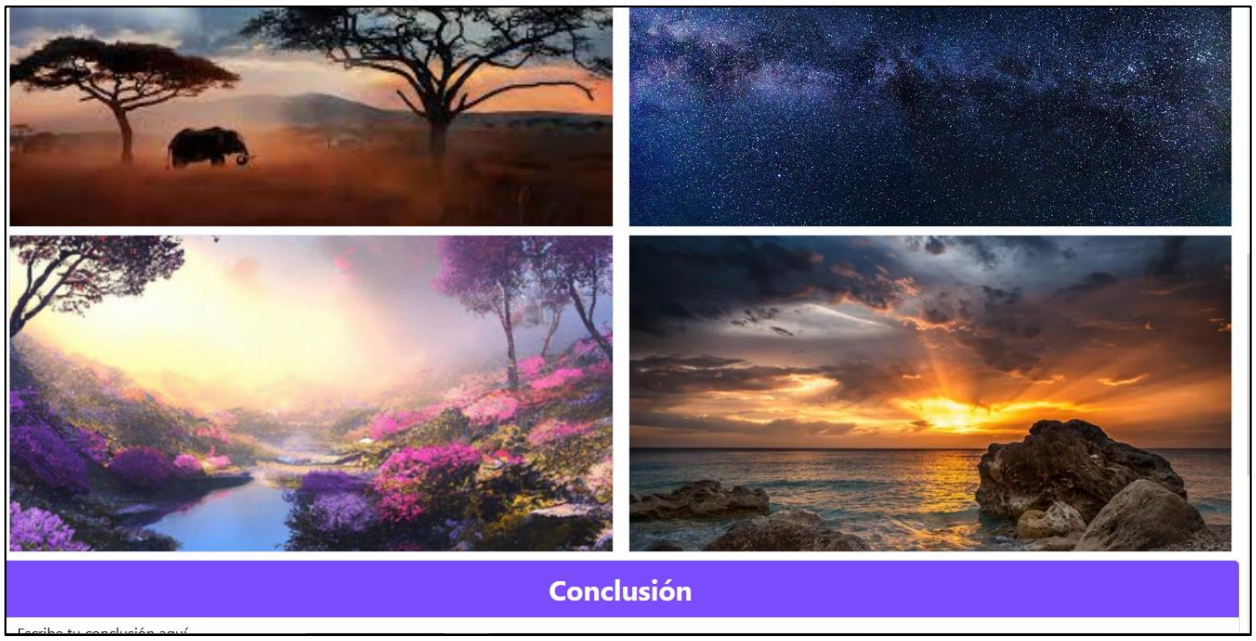


Imagen 73. Resultado de visualización para gráficas e imágenes de carrusel en el formato 3.

# Conclusiones

Este proyecto es el resultado de seis meses de servicio social en la Facultad de Medicina de la UNAM, y puedo concluir que el objetivo del cartel electrónico fue realizado, desde el desarrollo de la aplicación, hasta el proceso de despliegue de la aplicación productiva en *AppServ*, ya que permite hacer las funcionalidades esenciales como la edición, personalización y adición de contenido multimedia, así como el guardado de los datos del cartel, y el cambio de estatus dentro de su registro para que pueda ser en un futuro publicado de acuerdo a las condiciones que decidan los administradores. Por último, la visualización de un cartel para su vista previa es cuando se está editando y cuando la visualización ya se guardó en la base de datos.

Cabe resaltar que *Angular* permite integrar correctamente todos los componentes, siendo más modulares y creando una encapsulación para que a la hora de ejecutar un cambio en la programación sea sencillo realizarlo, en un solo componente y, este se repita en sus componentes padres; además de permitir una simplificación en el uso de servicios para que al momento de conectar con la *API* no sea tan robusto el proceso gracias a sus interfaces. Además, también gracias a sus módulos complementarios *Forms Module*, *Http Client Module* y *Routing Module*, ayudó a que la gestión de rutas de navegación, obtención e interpretación de resultados de la *API,* así como que formularios reactivos fueran más rápidos, por lo que fue acertado incluirlo como herramienta y más para un proyecto tan visualmente funcional como lo que es el cartel electrónico (*e-cartel*). Por último, la característica más usada de *Angular* son los componentes padres que escuchan a los hijos y viceversa. Para cuando el usuario ejecute un cambio con las herramientas, sea escuchado por parte ambos lados, este *two-way data binding* es una principal ventaja por la que se decidió usar *Angular* y no otro *framework*, sin embargo, no es descartable que también sea posible por medio de otro.

Aunque el cartel cumple con las funcionalidades principales, cabe recalcar que puede ser optimizado y mejorado aún más porque el usuario puede recuperarlo para volver a editar y no editar uno nuevo, así como un tablero que obtenga todos los *e-carteles* que se han creado y el usuario pueda eliminarlos o editarlo. Esto requeriría añadir un servicio más en *Angular*, para la eliminación de un cartel en la tabla de la base de datos. Con su *id* y crear su respectivo *endpoint* en la *API*, el servicio para obtener *e-cartel* ya está configurado y, permite previamente realizar la eliminación y modificación de uno antes guardado, y debe suponer una conexión junto con la tabla de usuarios y añadir otro campo a la tabla "carteles" para que estén enlazados. De la misma forma, se puede añadir una gestión de sesión para que los usuarios autorizados por el congreso sean los únicos que puedan acceder, esto sería realizado a través de los *Guard* que no son más que servicios de *Angular* dedicados a proteger las rutas una vez sea iniciada la sesión. Así mismo, este reporte sirve como documentación para ejecutar futuros cambios dentro del proyecto desarrollado en el servicio social y comprender cómo está estructurado, conocer las herramientas fueron usadas y sus versiones ocupadas.

Finalmente, respecto a mi desarrollo profesional, desde el *frontend*, *backend* y la conexión de la *API*, para la creación del *e-cartel*, resultó en realizar no solo varias investigaciones sino también en crear diferentes versiones que ayudaran a saber qué herramientas son las adecuadas y cuáles no eran óptimas para este proyecto. Tanto para los cambios reactivos que presenta la página de edición, que pudo ser resuelta por medio de herencia de variables, como el encontrar un elemento de entrada que permitiera controlar cambios de estilo: color, tamaño de fuente e integración de hipervínculos, como lo permite hacer el módulo de *TinyMce*, cuya funcionalidad es exactamente lo que se busca para personalizar sus apartados en cada sección. En resumen, el conocimiento que pude obtener, a través de todo lo anterior, se amplió más de lo que esperaba dado que este desarrollo fue una aplicación directa de los conocimientos obtenidos durante mi formación académica.

# Bibliografía


I. RxJS.(2023).RxJS - *API* List. RxJS. https://rxjs.dev/api

II. Tiny Technologies Inc.(2022).Tiny Docs. https://www.tiny.cloud/docs/

III. *PHP* Group.(2023).*PHP*: Documentation. https://www.php.net/docs.php

IV. *Angular*. (2023). *Angular* Docs. https://angular.io/docs

v. Xancopinca, A.(2022). Gaceta Facultad de Medicina. https://gaceta.facmed.unam.mx/index.php/2022/09/27/1er-congreso-internacional-de-educacion-en-ciencias-de-la-salud/

VI. Oriols, D. M. B., & Gutiérrez, J. G. (2020). *El gran libro de Angular*. Marcombo.