ESCUELA POLITÉCNICA NACIONAL

ESCUELA DE FORMACIÓN DE TECNÓLOGOS

APLICACIÓN WEB DE ILUSTRACIONES DIGITALES PARA ARTSPACE

TRABAJO DE INTEGRACIÓN CURRICULAR PRESENTADO COMO REQUISITO PARA LA OBTENCIÓN DEL TÍTULO DE TECNÓLOGO SUPERIOR EN DESARROLLO DE *SOFTWARE*

ROBERTO ANDRÉS CHACÓN LÓPEZ

DIRECTOR: JUAN PABLO ZALDUMBIDE PROAÑO

DMQ, marzo 2024

CERTIFICACIONES

Yo, **ROBERTO ANDRÉS CHACÓN LÓPEZ** declaro que el trabajo de integración curricular aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

ROBERTO ANDRÉS CHACÓN LÓPEZ

roberto.chacon@epn.edu.ec

robertochacon9514@hotmail.com

Certifico que el presente trabajo de integración curricular fue desarrollado por **ROBERTO ANDRÉS CHACÓN LÓPEZ**, bajo mi supervisión.

ING. JUAN PABLO ZALDUMBIDE

DIRECTOR

juan.zaldumbide@epn.edu.ec

DECLARACIÓN DE AUTORÍA

A través de la presente declaración, afirmamos que el trabajo de integración curricular aquí descrito, así como el (los) producto(s) resultante(s) del mismo, son públicos y estarán a disposición de la comunidad a través del repositorio institucional de la Escuela Politécnica Nacional; sin embargo, la titularidad de los derechos patrimoniales nos corresponde a los autores que hemos contribuido en el desarrollo del presente trabajo; observando para el efecto las disposiciones establecidas por el órgano competente en propiedad intelectual, la normativa interna y demás normas.

ROBERTO ANDRÉS CHACÓN LÓPEZ

DEDICATORIA

Dedicado a quienes son mi impulso para seguir adelante:

Dedicado a mi madre Janneth López, por ser mi impulso en esta vida, a mi padre Luis Chacón, por ayudarme a culminar mi etapa universitaria.

A mis hermanas Andrea Chacón y Anabel Chacón, siguán avanzando en lo que se propongan, verlas triunfar también es uno de mis objetivos.

A la memoria de mi abuelo Guillermo López (†) y mi tía Susana López (†)

Con mucho cariño para todos ellos.

¡Gracias por todo!

ROBERTO ANDRÉS CHACÓN LÓPEZ

AGRADECIMIENTO

Expreso mi gratitud más sincera a mi familia, cuyo apoyo inquebrantable ha sido mi mayor fortaleza a lo largo de este fascinante viaje académico. Su amor, paciencia y aliento constante han iluminado cada paso para cumplir este sueño.

Asimismo, quiero agradecer sinceramente a mis excelentes profesores. Su dedicación, conocimiento y orientación han sido fundamentales para mi progreso académico y personal. Cada lección y consejo que he recibido ha marcado mi educación.

Agradezco también a la prestigiosa universidad Escuela Politécnica Nacional por brindarme un entorno educativo fantástico. Mi formación integral ha sido significativamente mejorada por la calidad de la enseñanza y la riqueza de los recursos académicos.

Este logro no solo es mío, sino de todos quienes que han sido parte de mi vida universitaria. Mientras cierro este capítulo, mi corazón rebosa de gratitud porque llevo conmigo el amor de mi familia que me impulsarán hacia nuevos horizontes. ¡Gracias por participar en mi viaje y hacer posible este éxito!

ROBERTO ANDRÉS CHACÓN LÓPEZ

ÍNDICE DE CONTENIDO

RESUMEN

El objetivo del proyecto "Aplicación *web* de ilustraciones digitales para ArtSpace" es crear una página *web* segura y confiable donde los artistas pueden exhibir sus productos y los internautas puedan adquirir ilustraciones digitales de calidad. La aplicación tendrá un *frontend* y un *backend* desarrollados para brindar una experiencia de usuario atractiva e intuitiva.

Para facilitar la exploración de las ilustraciones digitales, se creó una interfaz de usuario moderna y estéticamente agradable con opciones de búsqueda avanzadas, sugerencias y filtros. El *backend* se encarga de garantizar el manejo del contenido multimedia, así como la gestión de usuarios, la autenticación y las políticas de la aplicación *web*.

Además, los artistas al subir su contenido multimedia en la aplicación *web*, la aplicación incorporará métodos de programación para clasificarlo, con la finalidad de mejorar la organización y en la búsqueda del contenido multimedia. Además, la aplicación web utiliza medidas de seguridad sólidas como la encriptación de datos y la autenticación de usuarios para proteger la seguridad y la privacidad de los usuarios.

El resultado final es una aplicación *web* donde los artistas pueden crear sus propios entornos virtuales, para mostrar sus ilustraciones digitales y recibir reseñas del contenido, mientras que los usuarios pueden explorar y descargar contenido multimedia. Se espera que la aplicación *web* fomente a la comunidad artística confianza entre vendedores y compradores, así mismo promover la autenticidad del arte digital en línea.

El proyecto se llevará a cabo utilizando la metodología SCRUM, lo que permitirá una adaptación flexible en el transcurso que se desarrolla el proyecto. Además, se proporcionarán pruebas detalladas para garantizar que la aplicación *web* funcione correctamente y se proporcionará una documentación completa para facilitar su uso y mantenimiento, con el objetivo de implementar nuevas funcionalidades a futuro.

PALABRAS CLAVE: ilustraciones digitales, *frontend*, *backend*, interfaz de usuario, autenticación, seguridad, SCRUM, experiencia de usuario.

ABSTRACT

The objective of the project "Digital Illustration Point of Sale Web Application for ArtSpace" is to create a secure and reliable website where artists can display their products and Internet users can purchase quality digital illustrations. The application will have a frontend and backend developed to provide an engaging and intuitive user experience.

To facilitate the exploration of digital artwork, a modern and aesthetically pleasing user interface was created with advanced search options, suggestions and filters. The backend is responsible for ensuring the management of multimedia content, as well as user management, authentication and system policies.

In addition, when artists upload their multimedia content to the web application, the system will incorporate programming methods to classify it, in order to improve the organization and search for multimedia content. Additionally, the web application uses strong security measures such as data encryption and user authentication to protect the security and privacy of users.

The end result is a web application where artists can create their own virtual environments, to display their digital artwork and receive content reviews, while users can explore and download multimedia content. The web application is expected to foster trust between sellers and buyers in the art community, as well as promote the authenticity of digital art online.

The project will be carried out using the SCRUM methodology, which will allow flexible adaptation over the course of the project. In addition, detailed testing will be provided to ensure that the system works correctly and comprehensive documentation will be provided to facilitate its use and maintenance, with the aim of implementing new functionalities in the future.

KEYWORDS: digital illustrations, frontend, backend, user interface, authentication, security, SCRUM, user experience.

1 DESCRIPCIÓN DEL COMPONENTE DESARROLLADO

El proyecto actual se enfoca en ofrecer espacios de comercialización para que los usuarios exhiban sus productos de ilustraciones digitales o contenido multimedia en general, una práctica que está ganando popularidad en el mundo actual. Lo mayoría de los artistas usan las redes sociales para comercializar sus productos y muchas veces existe estafas entre vendedores y compradores. También usan plataformas que no promueven los derechos de autor, provocando el *copyleft* en las ilustraciones digitales.

En la era digital, las obras de arte pueden ser compartidas y difundidas fácilmente a través de internet. Esto permite que los artistas lleguen de manera rápida y efectiva a una audiencia global. Los artistas ahora tienen más oportunidades de promocionar y vender sus obras gracias a las redes sociales y las plataformas de venta en línea [1]. El arte digital puede ser visualizado en un espacio virtual y a la vez compartido y experimentado en cualquier parte del mundo a través de Internet, a diferencia del arte tradicional, que se limita en un espacio físico. Además, el arte digital ha permitido a los artistas conectarse y colaborar en proyectos a través de plataformas en línea [1].

Las ventas y compras virtuales pueden presentar variedad de problemas que pueden ser causados por los usuarios o por el propio sistema. Los engaños y estafas son comunes en este campo, la vulnerabilidad también es un problema importante en la aplicación *web*, porque pueden afectar la confianza de los usuarios y poner en peligro la integridad del sistema. Además, la falta de autenticidad y verificación en el contenido de arte digital en línea puede existir dificultad en verificar la autenticidad de una ilustración digital, lo que provocaría desconfianza en los compradores. Los productos del intelecto humano ya sean científicos, artísticos o industriales, están protegidos por la propiedad intelectual. Los creadores, autores e inventores tienen un derecho temporal para prohibir a terceros utilizar el conocimiento que han creado [2].

Las reseñas de los usuarios en las publicaciones de los artistas ayudan en proteger los derechos de autor en las ilustraciones digitales y evita la reproducción no autorizada. También brinda una experiencia de usuario fluida y satisfactoria

además de la seguridad. Esto incluye una interfaz fácil de entender y un proceso de búsqueda simple. En un entorno virtual, la confianza entre compradores y vendedores es esencial. El éxito en una tienda de arte digital en línea depende de tener una imagen de marca sólida y una presencia en línea destacada. Aumenta la confianza de los compradores potenciales y establece una fuente confiable con buena reputación [3].

Para garantizar la confiabilidad, la aplicación web utiliza las reseñas de los comentarios de los usuarios para verificar la autenticidad de las ilustraciones digitales. La política de privacidad, también conocida como "Términos y Condiciones", es un documento o declaración específica emitida por un sitio web que describe las prácticas y procedimientos utilizados dentro de una página web para brindarle al usuario toda la transparencia sobre los datos que usa [4]. Los términos y condiciones de la aplicación web son cruciales para establecer las normas de uso del sistema y proteger los intereses tanto de los usuarios como del propietario del sitio web. Una sección de reseñas también se implementa en la aplicación *web* para generar confianza en los usuarios y que los artistas puedan recibir retroalimentación sobre la calidad y creatividad del servicio que brindan. Esto contribuirá en obtener usuarios confiables y tener una reputación positiva en la aplicación *web*.

Un entorno confiable y protegido para los usuarios se creará mediante la combinación de medidas de seguridad avanzadas, sistemas de verificación y opiniones de terceros. Se promoverá la autenticidad, la seguridad y la confiabilidad en el mercado de ventas de arte digital en línea al integrar estas soluciones.

1.1 Objetivo general

Crear una aplicación *web* segura y confiable, enfocado en productos sobre ilustraciones digitales, videos y *GIFs*, donde los artistas digitales puedan tener sus propios espacios comerciales para exhibir sus obras de arte digital. Los usuarios evitan fraudes y estafas al momento de adquirir contenido multimedia.

1.2 Objetivos específicos

1. OE1 Levantar de requerimientos de la aplicación *web*.
2. OE2 Implementar la arquitectura MVC de la aplicación *web* (*backend*).
3. OE3 Desarrollar de la interfaz de usuario (*frontend*)
4. OE4 Realizar pruebas de rendimiento, compatibilidad y de aceptación.

1.3 Alcance

El objetivo del proyecto es desarrollar una aplicación web que permita la comercialización de arte digital. Incluye diseños de interfaces y desarrollo funcional, así como la implementación de funcionalidades necesarias para permitir a los usuarios publicar *posts* y exhibir sus productos de arte digital. La aplicación *web* permite a los usuarios registrarse y crear sus perfiles. Los usuarios registrados pueden ingresar su información y administrar sus datos personales. Los usuarios podrán cargar imágenes, videos y *GIFs* en la aplicación *web*. La aplicación *web* proporciona herramientas para que los usuarios puedan describir sus contenidos, agregar etiquetas y establecer categorías para que su contenido tenga mayor facilidad en buscar. Los visitantes podrán navegar, buscar, explorar el contenido multimedia disponible. Se implementarán opciones de filtrado y categorización para facilitar la navegación por la interfaz y permitir a los usuarios encontrar contenido multimedia de su interés. Los usuarios podrán seleccionar obras de arte digital y contactarse con los compradores o vendedores para resolver cualquier problema o consulta. Se tomarán medidas para proteger los datos de los usuarios. Esto incluye encriptación de datos, políticas de privacidad claras y procedimientos de autenticación adecuados.

El proyecto no abarcará aspectos relacionados con el envío de productos físicos ni la gestión de pagos y comisiones entre los vendedores en línea. La aplicación *web* no contempla la creación ni producción de las ilustraciones o algún tipo de contenido multimedia. La aplicación *web* estará disponible únicamente en formato *web*, excluyendo la posibilidad de desarrollo de aplicaciones móviles. El proyecto no abordará aspectos legales y de propiedad intelectual relacionados con el contenido multimedia ofrecidos en la aplicación *web*. Los usuarios serán

responsables de cumplir los términos y condiciones sobre los derechos de autor en las ilustraciones digitales.

La aplicación *web* tendrá módulos que pueden ser accesibles y otros no, según el rol del usuario. Los roles posibles incluyen usuario (visitante), artista (*Blogger*) o administrador (*Admin*).

Inicio de sesión y registro: Sistema para el inicio de sesión y registro.

Perfil de usuario: Sistema para administrar los perfiles de los usuarios.

Contenido multimedia: Sistema para cargar y descargar imágenes, videos y *GIFs*.

Reseñas: Sistema de comentarios por usuarios (autentificados) en los *posts*.

Roles y permisos:

- Gestión de permisos de usuarios.
- Gestión de roles de usuarios.

1.4 Marco Teórico

1.4.1 Modelo MER

Modelo Físico

Representa la estructura de la base de datos, como entidades (tablas), relaciones entre entidades, claves primarias y foráneas [5]. Para facilitar la comprensión de la estructura de la base de datos, el modelo sirve como guía para la implementación, migraciones de datos y en la documentación técnica.

1.4.2 Interfaz de usuario

Principios de diseños

El diseño gráfico se enfoca principalmente en la comunicación visual y la creación de elementos gráficos que transmitan ideas y mensajes. Los diseñadores gráficos utilizan técnicas y herramientas para crear diseños que sean atractivos, efectivos y que cumplan con los objetivos del cliente [6]. Para obtener mayor facilidad de encontrar contenido multimedia, la ampliación *web* usa principios de diseños para que el usuario pueda navegar dentro de la aplicación *web*.

Diseño centrado en el usuario

Es una técnica que se enfoca en el cliente y sus demandas durante el proceso del diseño. Esto implica que la experiencia del usuario se considera en cada paso del proceso de diseño, desde el desarrollo hasta la implementación [7]. El DCU de la aplicación *web* mejora la experiencia del usuario, para que el usuario tenga mayor permanencia en el sitio *web*. Un DCU también puede mejorar la accesibilidad para los usuarios que usan dispositivos móviles.

Diseño gráfico

La combinación de elementos como imágenes, tipografía, colores y composición ayuda a comunicar mensajes visuales. Su objetivo principal es transmitir información de manera clara, efectiva y estética [8]. Gracias al diseño gráfico, el usuario pueda identificar y usar los elementos en la interfaz de la aplicación *web* como: menús de navegación, botones, enlaces, *input*, etc.

Accesibilidad

Es un principio fundamental en el diseño para garantizar que todos tengan acceso y oportunidades iguales, independientemente de sus habilidades o discapacidades [9]. La accesibilidad visual de la aplicación *web* se enfoca en desarrollar diseños que sean fáciles de leer y entender incluso para personas con discapacidad visual.

1.4.3 Desarrollo del *backend*

Arquitectura del *backend*

Modelo: Son clases que sirven para representar las tablas de la base de datos. Permite interactuar con la base de datos para realizar tareas como eliminar datos, insertar y actualizar [10].

Vista: En *Laravel* las vistas son archivos con extensión *Blade* que contienen código *HTML*, *CSS* y *JavaScript*. Las vistas permiten presentar información al usuario [11].

Controlador: Los controladores son clases encargadas de administrar la lógica de la aplicación y procesar las solicitudes del usuario [12].

Rutas: Las rutas permiten que la solicitud se enrute en el controlador deseado. La ruta Laravel más básica admite un identificador de activos uniforme (la ruta de tu ruta) y un cierre, que puede ser una función o una clase [13].

Vistas y *Blade* (motor de plantillas): *Laravel* utiliza plantillas o *templates*, que son los archivos principales y contienen segmentos de código que se repiten en varias vistas, como la barra de navegación y el menú de opciones [14]. El uso de plantillas permite reutilizar el código ahorrando tiempo en el desarrollo de la aplicación *web*.

Gestión de Datos

***Eloquent* ORM en *Laravel*:** Está incluido con *Laravel*, ofrece una implementación atractiva y fácil de usar cuando se trabaja con la base de datos. En la base de datos cada tabla tiene un "Modelo" que se usa para crear campos, relacionar tablas y asignar llaves primarias o foráneas en los campos [15].

Migraciones: Las migraciones permite cambiar el esquema de la base de datos y mantener actualizado el estado actual del esquema [16]. En el desarrollo, las migraciones con los *seeder* ayudan a llenar la base de datos con datos *faker* o datos de prueba.

Testing

Pruebas de compatibilidad en navegadores: Es fundamental realizar pruebas de compatibilidad en varios navegadores porque no todos funcionan de la misma manera. Un módulo de aplicación puede funcionar perfectamente en un navegador, pero puede no arrancar en otro [17]. Las pruebas de compatibilidad ayudan a verificar que todos los módulos de la aplicación *web* funcionen en distintos navegadores para que sea accesible para los usuarios.

Pruebas de rendimiento: Las pruebas de rendimiento utilizan pruebas de carga, estrés y estabilidad para evaluar el rendimiento de la aplicación *web* bajo una carga de trabajo [18].

Pruebas de aceptación: Las pruebas de aceptación se realizan al usuario para que utilice el software y verifica si cumple con sus expectativas. Por lo general estas pruebas se realizan antes que el producto salga a producción [19].

Despliegue

***Digital Ocean*:** Ofrece servicios de hospedaje en la nube personalizados para cada empresa, lo que permite a las empresas expandir su capacidad de hospedaje a medida que lo requieran [20].

Documentación del código

Uso de comentarios en código fuente: En el desarrollo del código fuente de la aplicación *web* se comentará cada fragmento de código para facilitar la comprensión de la funcionalidad.

Documentación automática con *phpDocumentor*: La guía de *phpDocumentor* se encuentra en bloques (*DocBlock)*. Estos bloques se colocan antes del elemento que están registrando [21].

1.4.4 Seguridad

Privacidad de datos

La capacidad de los usuarios de determinar quién puede acceder a sus datos personales y protegerlos frente a quienes no deberían tener acceso a ellos se conoce como privacidad de los datos [22]. *MySQL* brinda el encriptado de contraseñas cuando se almacenan en la base de datos.

Integridad de los datos

La integración de datos es un conjunto de técnicas y procedimientos comerciales que se utilizan para convertir datos de diferentes fuentes en datos confiables y valiosos [23]. La integración de MySQL en el proyecto proporciona mecanismos robustos para garantizar la integridad en los datos de la aplicación *web*.

Detención de ataques adversarios

Las medidas de seguridad de Laravel protegen las aplicaciones de vulnerabilidades como inyecciones *SQL*, ataques *CSRF* (*Cross-Site Request Forgery*) y *XSS* (*Scripting Cross-Site*) [24]. El uso del *framework Laravel* brinda protección contra inyección *SQL*, protección de rutas, *Middleware* de autenticación, bloqueos automáticos de usuarios después de un cierto número de intentos de inicio de sesión.

2 METODOLOGÍA

Se utilizará una metodología ágil al crear una aplicación web, específicamente la metodología *SCRUM*, para lograr un enfoque eficiente y flexible que permita adaptarse a los cambios y avances durante el proceso de desarrollo. *SCRUM* se ha seleccionado debido a su enfoque la planificación incremental y la capacidad de respuesta a los comentarios y requerimientos de los usuarios.

Para este proyecto se usa un tipo de trabajo exploratorio, se realiza investigación de herramientas y tecnologías para el desarrollo de la aplicación *web*. La creación de la aplicación *web* se basa en hipótesis tanto como el diseño, como en el funcionamiento. Se deduce los requisitos y funcionalidades basadas en las hipótesis planteadas, puesto que el proyecto es dirigido a la comunidad de artistas digitales en diferentes partes del mundo.

2.1 Metodología de Desarrollo

La gestión de un plan de desarrollo se conoce como metodología de desarrollo de software. Un procedimiento de desarrollo de programa aborda principalmente temas como la selección de propiedades para incluirlas en la versión actual, la fecha de lanzamiento del programa, quién trabaja en él y qué pruebas se realizan [32]. Seguidamente, se describen algunas de las cualidades de importancia en la metodología.

Roles

Un equipo *Scrum* generalmente consta de 3 a 9 miembros, junto con el *Scrum Master* y el *Product Owner*. Cada uno de estos puestos tiene tareas diferentes y debe rendir cuentas de manera diferente, tanto entre sí como para el resto de la organización. El equipo Scrum es la combinación de todos los roles [33]. Para el proyecto no se considera el *Product Owner,* el *Scrum Team* se divide en 2 roles:

Scrum Master

La función principal del *Scrum Master* es supervisar el método *Scrum* y ayudar a resolver problemas potenciales con la entrega del producto [33]. El *Scrum Master* tiene la obligación de hacer cumplir con todos los estándares de la metodología *Scrum*. El *Scrum Master* del proyecto se establece en la **TABLA I**

Development Team

Al final del ciclo de desarrollo, el equipo de desarrollo generalmente está formado por 3 a 9 profesionales que desarrollan el producto, organizan y gestionan para entregar un aumento de software [33]. En la **TABLA I** establece el *Development Team* del proyecto.

TABLA I: Asignación de responsabilidades.

ROLES	NOMBRES
Scrum Master	Ing. Juan Pablo Zaldumbide
Development Team	Roberto Andrés Chacón López

Artefactos

Los documentos e información llamados artefactos Scrum se crean y utilizan como parte del marco de trabajo Scrum. ofrecen claridad sobre el trabajo, el progreso y los objetivos del proyecto [34]. En el proceso de desarrollo de este proyecto, se utilizaron los siguientes artefactos:

Recopilación de Requerimientos

La recopilación de requisitos es el proceso de determinar las necesidades precisas del proyecto de principio a fin. Durante la fase inicial del proyecto, se lleva a cabo el proceso, pero se extiende a lo largo del proyecto [35]. La sección **Recopilación de requerimientos** del **ANEXO II** contiene todos los requisitos obtenidos después de las investigaciones y las necesidades de los internautas.

Historias de Usuario

La historia de usuario es una descripción general e informal de una función de software escrita desde la perspectiva del usuario final. Su objetivo es definir cómo brindará valor al cliente a través de una función de software. Es tentador creer que las historias de usuario son requisitos de la aplicación *web*. No obstante, no lo son [36]. Las **Historias de usuario** de acuerdo con los requisitos establecidos en el Error! Reference source not found., **TABLA I**, se presenta un ejemplo de la **TABLA II**.

TABLA II: Historias de usuarios 006 – Gestión de roles y mantenimiento de datos.

HISTORIAS DE USUARIO (HU)	
Identificador: HU006	**Tipo de usuario:** Administrador (*Admin*)
Nombre de la historia: Gestión de roles y mantenimiento de datos	
Prioridad: Alta	**Riesgo en desarrollo:** Alta
Número de iteración: 1	
Responsable: Roberto Chacón	
Descripción: Un usuario administrador (*Admin*) establece el tipo de rol y permisos, así mismo eliminar los roles asignados. También necesita modificar y eliminar datos, para que la aplicación *web* tenga mejor rendimiento con la información. • Los administradores pueden asignar el rol *Admin* a los usuarios. • Los administradores pueden asignar el rol *Blogger* a los usuarios. También puede agregar categorías y etiquetas para que los usuarios asignen a sus *posts*: • Tipo de categoría. • Tipo de etiqueta. Además, puede visualizar la lista de usuarios y poder modificarlos: • Borrar el perfil de usuario. • Asignar un rol al usuario. • Quitar roles al usuario.	
Observación: En el panel de control del administrador existe una opción para poder verificar el contenido y determinar si es aprobado o no.	

Product Backlog

Los requerimientos, casos de uso, tareas y dependencias son parte del inventario de productos. Es la fuente principal de información sobre el producto en *Scrum*, una lista en cualquier formato que contiene todos los requisitos que se debe

implementar en el producto [37]. El ***Product Backlog*** del proyecto se encuentra en el **ANEXO II.**

Sprint Backlog

Se trata de una lista de actividades que se debe completar en cada *Sprint*. Estos componentes suelen incluir tareas técnicas que permiten una mayor calidad de software [37]. El **ANEXO II** contiene información detallada sobre el ***Sprint Backlog***.

2.2 Diseño de interfaces

Los *mockups* son montajes que hacen los diseñadores gráficos y los diseñadores *web* para mostrar a sus clientes cómo se imprimirán sus diseños en una superficie [38]. En el diseño de los prototipados de las interfaces se usó la herramienta *Framer* que permite la creación de las interfaces de la aplicación *web* en versión *web*.

Framer

Framer permite crear prototipos tanto para la interfaz de usuario (*UI*) como la experiencia de usuario (*UX*) [26]. El diseño de los prototipos se encuentra en **ANEXO II**, **Diseño de interfaces**. Un ejemplo del prototipo de la página principal (*home*) muestra en la **Figura 2.1**.

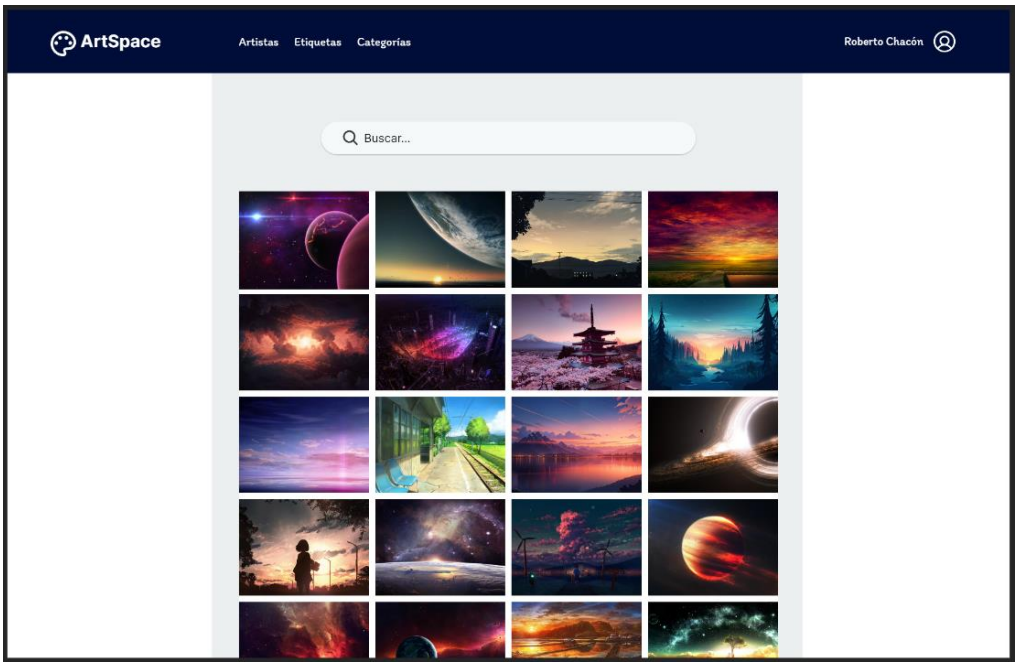


Figura 2.1: Página principal de la aplicación *web* diseñado en *Framer.*

Herramientas utilizadas para las interfaces

Cada herramienta para maquetar las vistas de la aplicación *web* se detalla en la **TABLA III**.

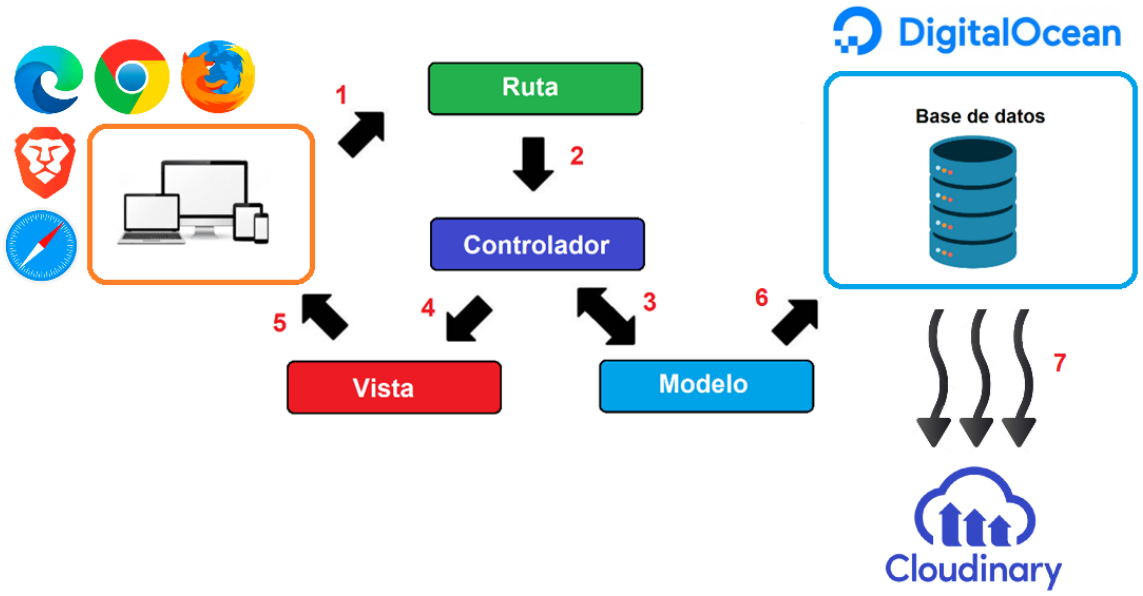
TABLA III: Herramientas de desarrollo para el *Frontend*.

Herramientas	Uso en el desarrollo
Framer	*Software* para realizar diseños y prototipado.
Jetstream	Ofrece una interfaz y funcionalidad en el inicio y registro de sesión, también en crear interfaces para el perfil de usuario.
Livewire	Ofrece componentes en las interfaces para el desarrollo de menús de navegación, filtros de búsqueda en tiempo real.
AlpineJS	Ofrece componentes personalizados en la creación de paginación, *scroll*, ventanas emergentes.
Tailwind	Ofrece estilos personalizados en diseñar tablas para el manejo de CRUD sin la necesidad de escribir código *CSS*.
AdminLTE	Ofrece plantillas de diseño para crear paneles de administración.

2.3 Diseño de la arquitectura

Patrón arquitectónico

El patrón arquitectónico distingue la lógica comercial (*Backend*) de la interfaz de usuario (*Frontend*), permitiendo la interactividad por separado, en ambos casos aumenta la flexibilidad y la reutilización de las aplicaciones [39]. El patrón arquitectónico que se usa en el proyecto se muestra en la **Figura 2.2**



1. El usuario envía una petición desde dispositivos: móviles, tables o computador.
2. Se direcciona la petición al controlador.
3. Interactuar con el modelo de datos.
4. El resultado se presenta en la vista.
5. Renderiza la vista en el navegador.
6. Los datos se guardan en la base de datos en ***DigitalOcean***.
7. Las imágenes y videos se guardan en ***Cloudinary*** mediante ***DigitalOcean***.

Figura 2.2: Patrón arquitectónico del proyecto.

2.4 Herramientas de desarrollo

Las herramientas que muestra en la **TABLA IV** permiten que el desarrollo de la aplicación *web* sea rápido y eficiente.

TABLA IV: Herramientas para el desarrollo de la aplicación *web.*

HERRAMIENTAS	JUSTIFICACIÓN
PowerDesigner	Para combinar análisis de negocios con soluciones formales de diseño de base de datos, *PowerDesigner* combina una amplia gama de técnicas de modelización [25]. *PowerDesigner* facilita la visualización detallada del modelo físico.
Framer	*Framer* permite crear prototipos tanto para la interfaz de usuario (*UI*) como la experiencia de usuario (*UX*) [26]. Los

	diseños de las interfaces de usuarios se desarrollan en *Framer.*
MySQL	*MySQL* organiza la información en tablas relacionadas entre sí [27]. *MySQL* sirve para administrar los datos tanto de manera local como en producción del proyecto.
Laravel	*Laravel* es un marco de aplicaciones *web* poderoso y sofisticado que utiliza la arquitectura modelo, vista y controlador (MVC) para facilitar el desarrollo de aplicaciones *web* [28]. El uso del *framework* facilita en desarrollar los entornos de trabajo para el *frontend* y *backend* en la aplicación *web*.
Visual Studio Code	*Visual Studio Code* es un editor de código fuente. Cuenta con extensiones para facilitar la depuración del código y ejecuta código en cualquier lenguaje de programación [29].
XAMPP	El servidor *web* local *XAMPP* permite la crear y probar páginas *web* u otros componentes de programación [30]. Con la herramienta *XAMPP* se crea de un propio servidor local y evita conflictos con el uso de plantillas en el *frontend*.
Digital Ocean	*Digital Ocean* ofrece opciones de hospedaje en la nube superiores a los métodos convencionales [31]. *Digital Ocean* será el *hosting* para desplegar el proyecto.

3 RESULTADOS

Los resultados de cada *Sprint* se muestran en esta sección.

3.1 *Sprint* 0. CONFIGURACIÓN DEL AMBIENTE DE DESARROLLO

Las tareas para el *Sprint* 0 son:

- Levantar requerimientos.
- Determinar roles de usuarios.
- Instalación de herramientas y Configuración.
- Creación de un nuevo proyecto en *Laravel*.

- Configuración de *Composer*.

Levantar requerimientos

En **ANEXO II**, **Tabla I** se presenta los requerimientos que debe cumplir en la aplicación *web*.

Determinar roles de usuarios

La aplicación *web* tiene roles como: Usuarios visitantes, usuarios *Blogger* y administrador, para saber en detalle el papel que desempeña cada uno de los roles, en **ANEXO II**, **Historias de usuarios** se especifica las actividades.

Instalación de herramientas y Configuración

Se utiliza herramientas de depuración de código para el desarrollo de la aplicación *web*.

- Se usa la herramienta *Visual Studio Code* en el desarrollo del proyecto. Este *IDE* permite usar extensiones para ahorrar tiempo en editar código y que la producción sea más rápida. En la **Figura 3.1** muestra una de las varias extensiones que se usa en el desarrollo. La extensión *Laravel Blade Snippets* permite escribir código *Blade* de manera más rápida y efectiva.

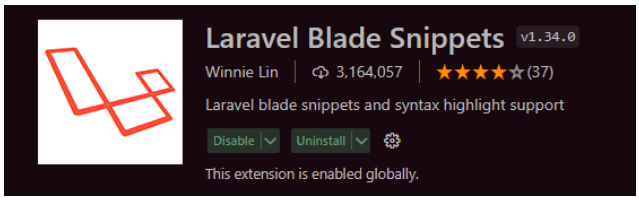


Figura 3.1: Extensión *Laravel Blade Snippets.*

- Se requiere la herramienta *XAMPP* para crear un entorno de desarrollo de manera local. También permite gestionar la base de datos *MySQL*. El panel de control que se utiliza para administrar la base de datos (*MySQL*) se muestra en la **Figura 3.2**.

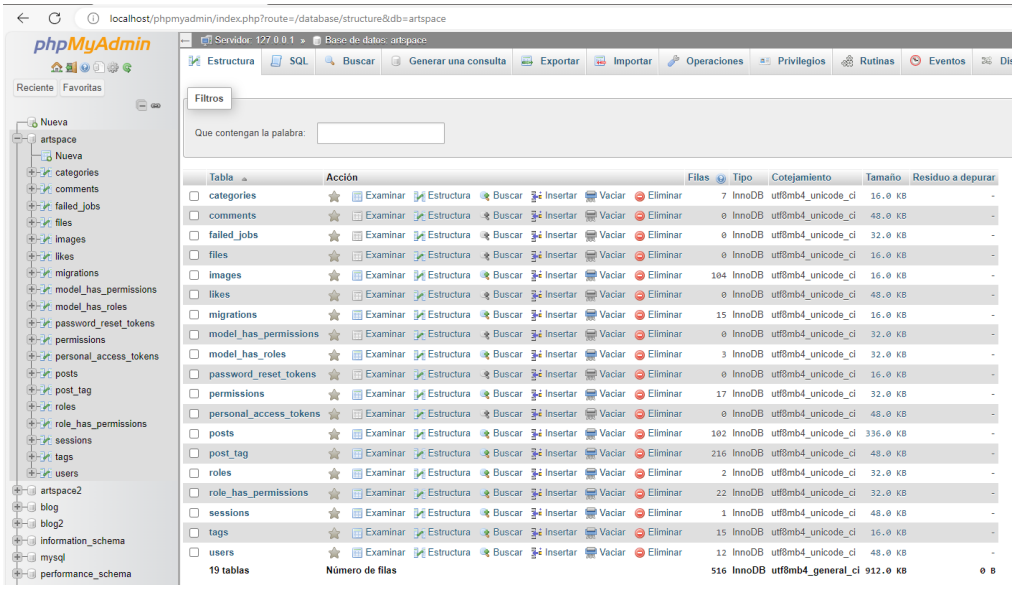


Figura 3.2: Panel de control *phpMyAdmin.*

- La Instalación de *Composer* en el *framework Laravel*, permite manejar los paquetes para el lenguaje de programación *PHP*. Finalizada la instalación en la **Figura 3.3** muestra el archivo de configuración (*composer.json*) para administrar las dependencias de *PHP*.

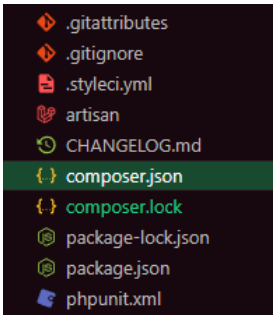


Figura 3.3: Archivo de configuración *composer.json.*

Crear de un nuevo proyecto en *Laravel*

Cuando se instala *XAMPP* también incluye *PHP*, se debe verificar que versión de *PHP* se está usando. Se ejecuta *XAMPP Control Panel*, se inicia *Apache* y en *Amin* se puede navegar por el sitio de *Apache*, en *PHPinfo* se observa la versión de *PHP* como muestra en la **Figura 3.4.**

En la ruta: **C:\xampp\htdocs** del sistema operativo *Windows*, será el sitio donde se crea el proyecto. El directorio principal del servidor *web Apache* es *htdocs*. Con la

url **http://localhost/nombre_del_proyecto** facilita el acceso del proyecto *Laravel* a través del navegador *web*.

PHP Version 8.2.4

System	Windows NT LAPTOP-AE8OE37F 10.0 build 19044 (Windows 10) AMD64
Build Date	Mar 14 2023 17:50:26
Build System	Microsoft Windows Server 2019 Datacenter [10.0.17763]
Compiler	Visual C++ 2019
Architecture	x64
Configure Command	cscript /nologo /e:jscript configure.js "--enable-snapshot-build" "--enable-debug-pack" "--with-pdo-oci=..\..\..\..\instantclient\sdk,shared" "--with-oci8-19=..\..\..\..\instantclient\sdk,shared" "--enable-object-out-dir=../obj/" "--enable-com-dotnet=shared" "--without-analyzer" "--with-pgo"
Server API	Apache 2.0 Handler
Virtual Directory Support	enabled
Configuration File (php.ini) Path	*no value*
Loaded Configuration File	C:\xampp\php\php.ini
Scan this dir for additional .ini files	(none)
Additional .ini files parsed	(none)
PHP API	20220829
PHP Extension	20220829
Zend Extension	420220829
Zend Extension Build	API420220829,TS,VS16
PHP Extension Build	API20220829,TS,VS16
Debug Build	no
Thread Safety	enabled
Thread API	Windows Threads
Zend Signal Handling	disabled
Zend Memory Manager	enabled
Zend Multibyte Support	provided by mbstring
IPv6 Support	enabled
DTrace Support	disabled
Registered PHP Streams	php, file, glob, data, http, ftp, compress.zlib, compress.bzip2, https, ftps, phar
Registered Stream Socket Transports	tcp, udp, ssl, tls, tlsv1.0, tlsv1.1, tlsv1.2, tlsv1.3
Registered Stream Filters	convert.iconv.*, string.rot13, string.toupper, string.tolower, convert.*, consumed, dechunk, zlib.*, bzip2.*

Figura 3.4: Servidor *web Apache*.

Después se crea una nueva terminal de *Windows* y se usa el comando **C1**, **TABLA X**, como se visualiza en la **Figura 3.5.**

```
C:\xampp\htdocs\LARAVEL>laravel new ArtSpace

   Creating a "laravel/laravel" project at "./ArtSpace"
   Installing laravel/laravel (v10.2.10)
       Failed to download laravel/laravel from dist: The zip extension and u
.
The php.ini used by your command-line PHP is: C:\xampp\php\php.ini
    Now trying to download from source
  - Syncing laravel/laravel (v10.2.10) into cache
      - Installing laravel/laravel (v10.2.10): Cloning d6a2d8b837 from cache
    Created project in C:\xampp\htdocs\LARAVEL/ArtSpace
> @php -r "file_exists('.env') || copy('.env.example', '.env');"
    Loading composer repositories with package information
    Updating dependencies
    Lock file operations: 111 installs, 0 updates, 0 removals
  - Locking brick/math (0.11.0)
  - Locking carbonphp/carbon-doctrine-types (3.1.0)
  - Locking dflydev/dot-access-data (v3.0.2)
```

Figura 3.5: Creación del proyecto “ArtSpace”.

En la carpeta raíz del proyecto **C:\xampp\htdocs\LARAVEL\ArtSpace**, se puede acceder en la termina de *Visual Studio Code* o en la terminal de *Windows*, en la **Figura 3.6** se observa el comando “cd” para ingresar al proyecto.

```
C:\xampp\htdocs\LARAVEL>cd ArtSpace
```

Figura 3.6: Carpeta raíz del proyecto.

La estructura de carpetas del proyecto se observa en la **Figura 3.7**.

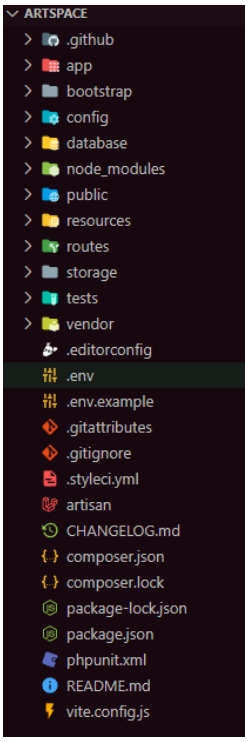


Figura 3.7: Estructura de carpetas del proyecto.

Para verificar que el proyecto se creó, se abre la carpeta raíz en *Visual Studio Code* y también se inicia el programa *XAMPP*, después con el comando **C2, TABLA X,** el proyecto se abre en un navegador. La interfaz de *Laravel* (*welcome*) se presenta en la **Figura 3.8**.

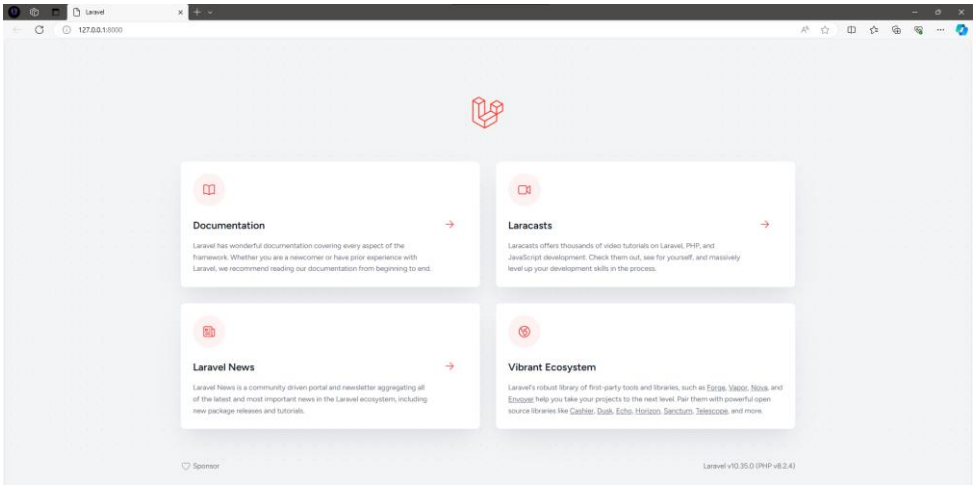


Figura 3.8: Proyecto ejecutado en el navegador.

Se usa el comando **C3, TABLA X,** para instalar *Jetstream*. Es necesario tener instalado *Laravel* 8 para que el comando funcione, refrescamos la página y se genera un apartado para iniciar sesión y para registrar como se presenta en la **Figura 3.9**. Cuando se instala *Jetstream* también se instala *Alpine. Alpine* permite agregar funcionalidad en componentes de *Jetstream.*

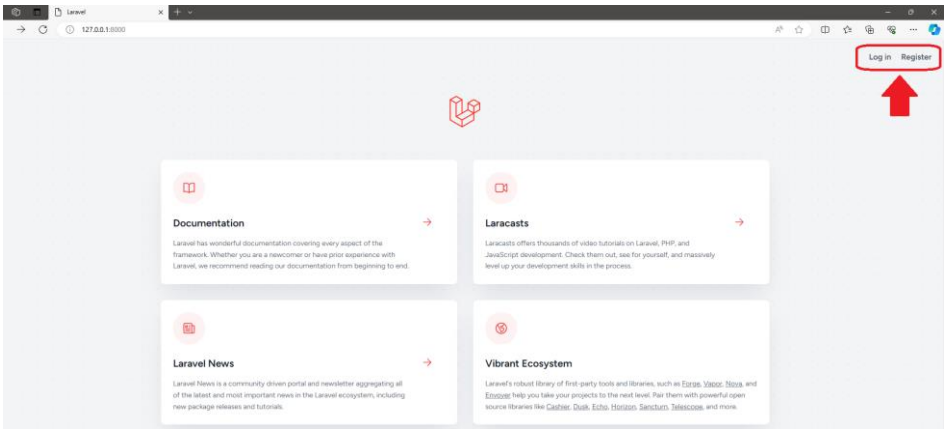


Figura 3.9: Instalación de *Jetstream*.

Después de instalar *Jetstream* con el comando **C4**, **TABLA X** se instala las dependencias y configuraciones de *Jetstream*.

Para que en la aplicación no produzca errores, se realiza la migración de las tablas que se generan por defecto en el *framework,* para ejecutar las migraciones se usa el comando **C5, TABLA X.**

Con el comando **C6, TABLA X** se instala el manejador de paquetes de *node* (npm), contiene paquetes de JavaScript que permite instalar componentes que se ejecutan del lado del cliente (*frontend*) y con el uso del comando **C7, TABLA X** permite correr npm.

En la **Figura 3.10** se observa que por defecto se nombra a la base de datos con un nombre similar al nombre del proyecto, esa sección de encuentra en el archivo **env** de la estructura del proyecto.

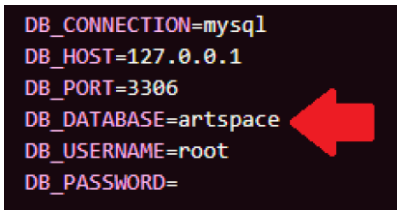


Figura 3.10: Archivo env.

Ahora se ingresa a *MySQL* y se crea una nueva base de datos, esta base de datos debe tener el mismo nombre que contiene el archivo **env** (*DB_DATABASE*) como se observa en la **Figura 3.11**.

Figura 3.11: Creación de nueva base de datos en *MySQL*.

Configuración de *Composer*

Se ubica en la raíz del proyecto, el comando **C8**, **TABLA X** permite instala los paquetes en el proyecto, solo se ejecuta una única vez el comando **C8, TABLA X**.

3.2 *Sprint* 1. ARQUITECTURA MVC

Las tareas para el *Sprint* 1 son:

- Creación de tablas y modelos.
- Establecer relaciones entre las tablas en los modelos.
- Crear *factories* (datos falsos) para pruebas en la conexión a *MySQL.*
- Generar un dominio local.
- Creación del menú *responsive* con *Tailwind* y *Alpine.*

Creación de tablas y modelos

Para crear las tablas del proyecto primero se crea las entidades (tablas) fuertes, las entidades fuertes no dependen de otras entidades (entidades: *categories, tags, images*).

- **Entidad *Category***: El comando **C9**, **TABLA X**, se usa para crear el modelo *Category* con su respectiva migración.
- **Entidad *Post*:** El comando **C10**, **TABLA X**, se usa para crear el modelo *Post* con su respectiva migración.
- **Entidad *Tag*:** El comando **C11**, **TABLA X**, se usa para crear el modelo *Tag* con su respectiva migración.
- **Tabla intermedia *post_tag*:** El comando **C12**, **TABLA X**, se usa para crear la tabla intermedia *post_tag*.
- **Entidad *Image*:** El comando **C13**, **TABLA X**, se usa para crear el modelo *Image* con su respectiva migración.

Establecer relaciones entre tablas en los modelos

Después de crear las tablas se procede a establecer las relaciones, a continuación, se explica la lógica de la relación entre tablas:

- **Tablas *users* – *posts* (relación de uno a muchos):** Un usuario podrá subir varios *posts* y un *post* solo le pertenece a un solo usuario:

- **Tablas *categories* – *posts* (relación de uno a muchos):** Dentro de una categoría existen varios *posts* y un *post* solo le pertenece a una categoría.
- **Tablas *posts* – *tags* (relación de muchos a muchos):** Un *post* puede tener varias etiquetas y una etiquete puede estar en varios *posts*.
- **Tabla *post_tag* (tabla intermedia):** Cuando exista una relación de muchos a muchos debemos crear una tabla intermedia, en este caso la tabla intermedia se llama *post_tag.*
- **Tabla *images* (relación polimórfica):** Se almacena todas las *urls* de las imágenes y videos. En el campo "*imageable_id*" se coloca el *id* con la entidad que queramos relacionar y en el campo "*imageable_type*" se ubica en el modelo de la entidad que se quiere relacionar.
- **Tabla *comments* (relación polimórfica):** Se almacena todos los comentarios de los usuarios.

Una vez creada las tablas con sus respetivas relaciones, en la **Figura 3.12** se puede observar las entidades generadas por *MySQL*, es importante mencionar que el *framework Laravel* genera otras entidades como: *password_reser_tokens, sessions, failed_jobs, migrations* y *personal_access_tokens*, estas entidades no van a hacer usadas, las entidades como: *categories, users, posts, tags, comments, post_tag* e *images* son tablas que se usa para administrar los datos de la aplicación *web*. Las entidades: *roles, model_has_roles, permissions, model_has_permissions* y *role_has_permissions*, son tablas generadas por el paquete de *Laravel permission*, se usa para administrar el sistema de roles y permisos.

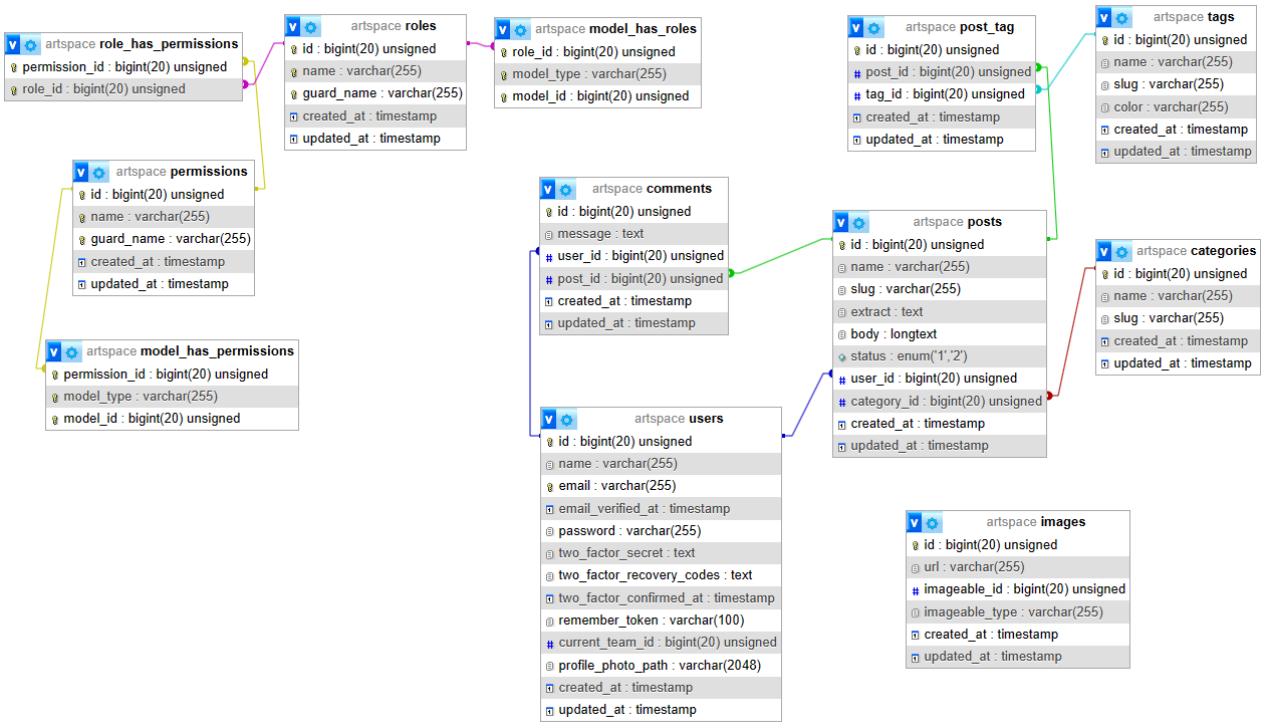


Figura 3.12: Entidades generadas por *MySQL*.

Crear *factories* (datos falsos) para pruebas en la conexión con *MySQL*

En la terminal de *Visual Studio Code* se ejecuta los comandos **C14**, **C15**, **C16** y **C17**, **TABLA X**, para crear los *factories* de las tablas *categories, posts, tags y images* respectivamente, para la tabla *user* no se crea *factories* porque el *framework* lo crea automáticamente. El uso de los *factories* permite usar datos falsos para realizar pruebas en el manejo de datos. En la **Figura 3.13** se presenta los *fatories* de *category, image, post, tag* y *user*.

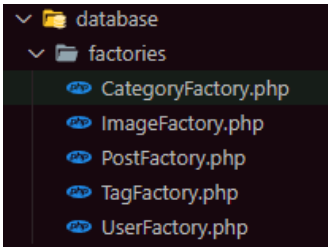


Figura 3.13: Creación de *factories*.

Se puede ver un ejemplo de la creación los datos falsos del *factories PostFactory* en la **Figura 3.14.**

```php
class PostFactory extends Factory
{
    /**
     * Define the model's default state.
     *
     * @return array<string, mixed>
     */
    public function definition(): array
    {
        $name = $this->faker->unique()->sentence();

        return [
            'name' => $name,
            'slug' => Str::slug($name),
            'extract' => $this->faker->text(250),
            'body' => $this->faker->text(2000),
            'status' => $this->faker->randomElement([1, 2]),
            'category_id' => Category::all()->random()->id,
            'user_id' => User::all()->random()->id,
        ];
    }
}
```

Figura 3.14: *factories PostFactory*.

Para que se generen los datos falsos se debe crear los *seeders* con los comandos **C18** y **C19**, **TABLA X,** en la **Figura 3.15** muestra los *seeders* creados.

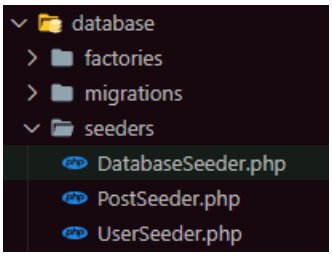


Figura 3.15: Creación de *seeders*.

En la **Figura 3.16** muestra un ejemplo de cómo instruimos en el *seeder UserSeeder* para crear las credenciales del *admin* y 80 registros de usuarios falsos.

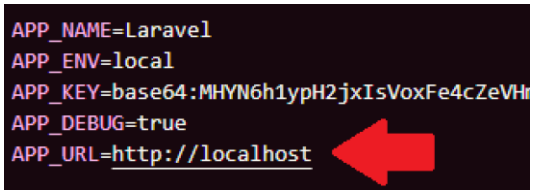
```php
class UserSeeder extends Seeder
{
    /**
     * Run the database seeds.
     */
    public function run(): void
    {
        // Creo mis credenciales
        User::create([
            'name' => 'Roberto Chacon',
            'email' => 'rch@gmail.com',
            'password' => bcrypt('123456789'),
        ]);

        // crear los registros falsos
        User::factory(80)->create(); // 80 registros falsos
    }
}
```

Figura 3.16: *Seeder UserSeeder*.

Con el comando **C20, TABLA X** elimina las tablas existentes y los registros después nuevamente crea las tablas y realiza las migraciones.

Generar un dominio local

Cuando se quiera recuperar una imagen, el *framework* por defecto hace que todas las imágenes se encuentran en el dominio *localhost*, en la **Figura 3.17** muestra el archivo **env** la *url*.



```
APP_NAME=Laravel
APP_ENV=local
APP_KEY=base64:MHYN6h1ypH2jxIsVoxFe4cZeVH
APP_DEBUG=true
APP_URL=http://localhost
```

Figura 3.17: Archivo **env**.

Para poder mostrar las imágenes en el sitio *web* de forma local, se usa la dirección **artspace.ec**, a continuación, se indica como generar un dominio local:

1. Se ubica en la ruta **C:\Windows\System32\drivers\etc\hosts** del sistema operativo *Windows*, ejecutamos el *bloc* de notas en modo administrador y se agrega el dominio, como muestra en la **Figura 3.18**.

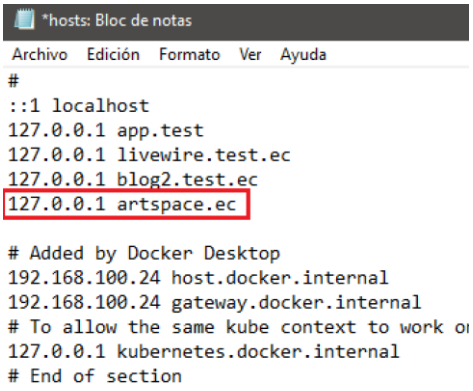


Figura 3.18: Dominio local.

2. Se dirige a la carpeta donde se instaló *XAMPP*, la ruta **C:\xampp\apache\conf\extra\httpd-vhosts.conf** se debe agrega una única vez el fragmento de código que presenta en la **Figura 3.19**.

```
NameVirtualHost *
<VirtualHost *>
  DocumentRoot "C:\xampp\htdocs"
  ServerName localhost
</VirtualHost>
```

Figura 3.19: Dominio local.

Después de agregar los comandos de la **Figura 3.19,** en la **Figura 3.20**. muestra los comandos que se debe añadir.

```
<VirtualHost *>
  DocumentRoot "C:\xampp\htdocs\LARAVEL\ArtSpace\public"
  ServerName artspace.ec
  <Directory "C:\xampp\htdocs\LARAVEL\ArtSpace\public">
    Require all granted
  </Directory>
</VirtualHost>
```

Figura 3.20: Dominio local.

3. Por último, en el archivo **env** se realiza el cambio de la dirección (*APP_URL*), como se puede observar en la **Figura 3.21**.

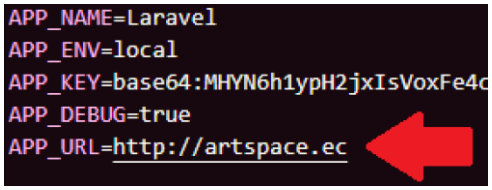


Figura 3.21: Dominio local.

Creación del menú responsive con Tailwind, Alpine y Livewire

En la **Figura 3.22** muestra un componente creado por desarrolladores de *Tailwind*, se puede conseguir en la página oficial de *Tailwind* (documentación). Este componente se integra como platilla para desarrollar el menú de navegación del proyecto.

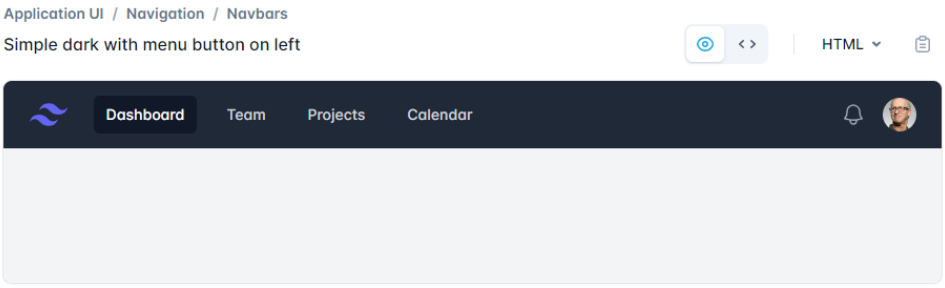


Figura 3.22: Componente *Tailwind*.

Se crea un componente de *Livewire* con el comando **C21**, **TABLA X**, al usar este comando crea un archivo *PHP* y un archivo *Blade* en la ruta **app\Livewire\Navigation.php** del *framework* como se observa en la **Figura 3.23**. El archivo *PHP* (Navigation.php) renderiza el contenido del archivo *Blade* (navigation.blade.php).

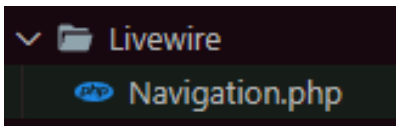


Figura 3.23: Componente *Livewire*.

En la **Figura 3.24** muestra el archivo *Blade* en la ruta **resources\views\livewire\navigation.blade.php,** en este componente se desarrolla los estilos con *Tailwind* y *Alpine.*

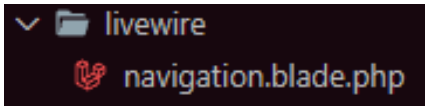


Figura 3.24: Componente *Livewire*.

En la ruta **resources\views\layouts\app.blade.php** del *framework* se encuentran los diseños de la interfaz de inicio de sesión, como presenta la **Figura 3.25**, en este archivo se modifica el estilo de acuerdo con el prototipado que se diseñó.

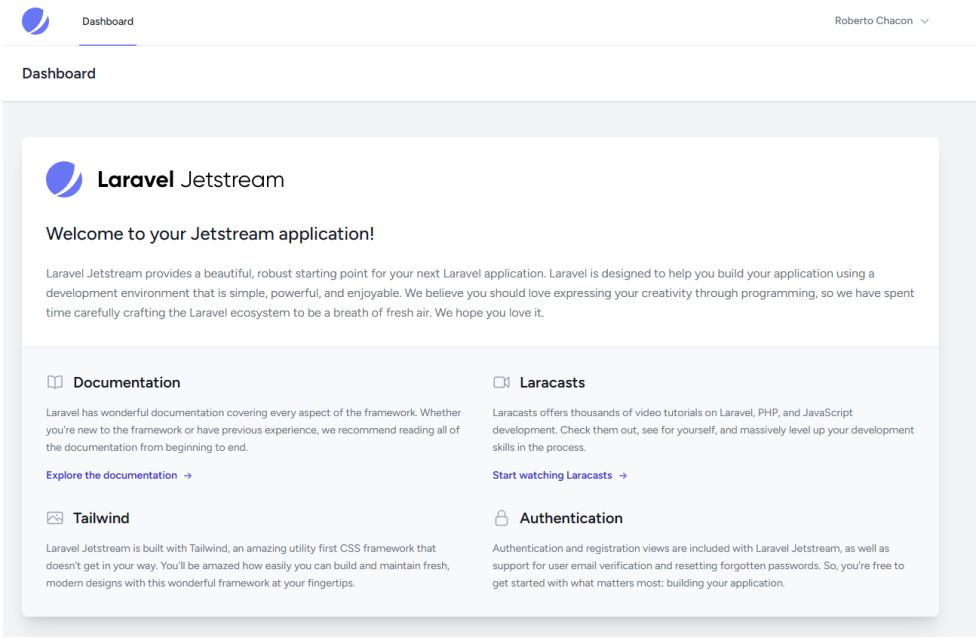


Figura 3.25: Componente *Tailwind*.

3.3 *Sprint* 2. MÓDULO PARA USUARIOS/USUARIOS VISITANTE

Las tareas para el *Sprint* 2 son:

- Diseño e implementación: Inicio de sesión, registro y perfil de usuario con el uso del paquete de *Jetstream.*
- Mostrar listado y detalle de *posts.*
- Instalación de *AdminLTE 3* para el uso de la plantilla panel de administrador.
- CRUD's de categorías, etiquetas y *posts*.
- Visualización de formularios y validaciones.

- Reglas de autorización.

Diseño e implementación: Inicio de sesión, registro y perfil de usuario con el uso del paquete de *Jetstream*

En el *Sprint* 1 se instaló el paquete de *Jeststream* con **el C4**, **TABLA X**, este paquete ofrece interfaces para el inicio de sesión y registro, como muestra en la **Figura 3.26**.

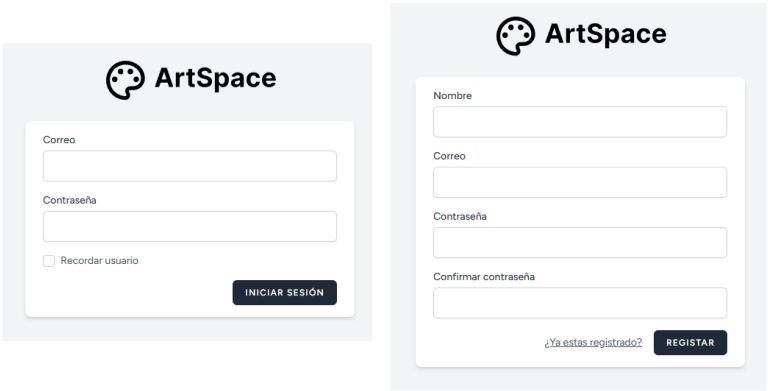


Figura 3.26: Inicio de sesión.

Así mismo el paquete de *Jeststream* genera una interfaz de perfil de usuario para que el usuario pueda modificar su correo, nombre de usuario, contraseña, cerrar su cuenta en otros navegadores y borrar su cuenta, como muestra en la **Figura 3.27**. Cabe mencionar que se cambió los estilos de acuerdo con el prototipado.

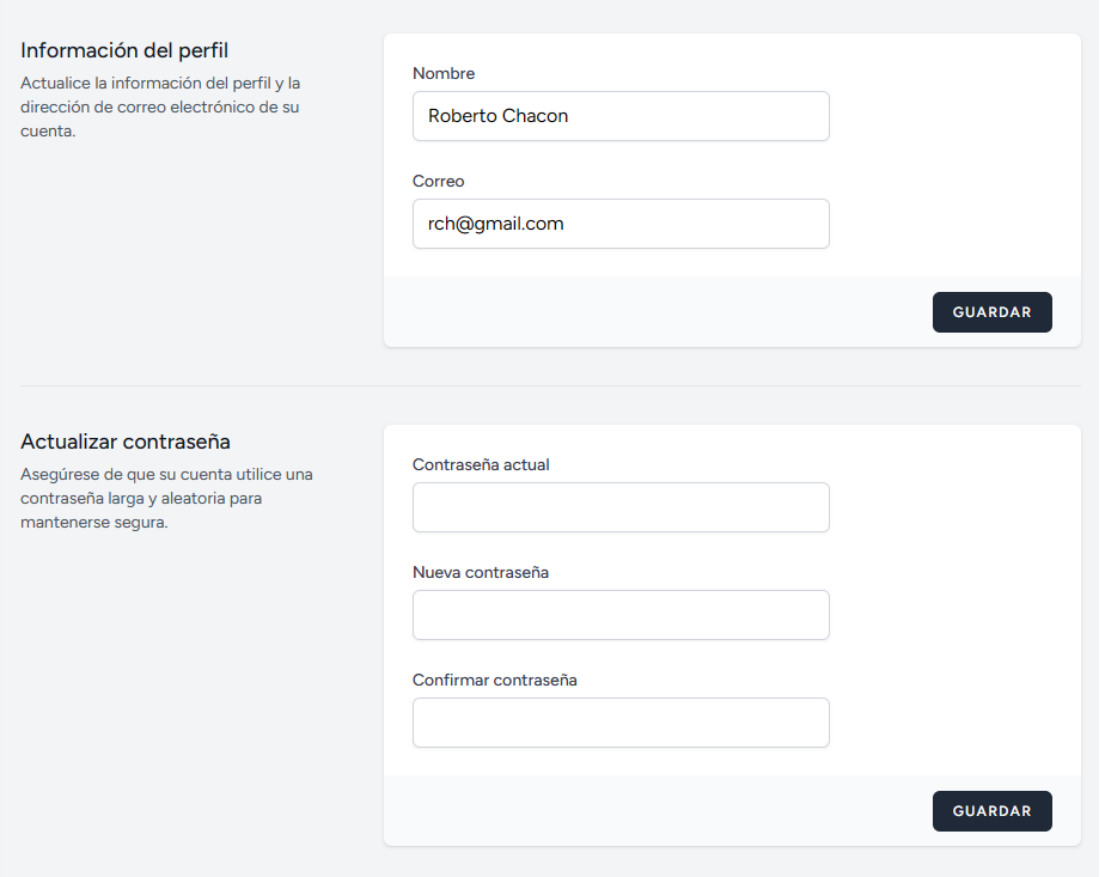


Figura 3.27: Perfil de usuario.

Mostrar listado y detalle de *posts*

Se necesita de un controlador para traer los datos de la base de datos, para generar un controlador se usa el comando **C22**, **TABLA X**, para crear el controlador *PostController*, este controlador tendrá métodos para rescatar la información de la base de datos como las etiquetas y categorías de los *posts*. En la **Figura 3.28** muestra un ejemplo de traer los posts.

```php
public function index(){
        $posts = Post::where('status', 2)->latest('id')->paginate(30); // paginate(50): numero de posts que se visualiza

        return view('posts.index', compact('posts'));
    }
```

Figura 3.28: Vistas *Blade*.

También se necesita vistas para mostrar los datos al usuario para lo cual se crea cuatro vistas en la ruta **resources\views\posts**, la vista *index* muestra las

imágenes en la página principal (*home*), la vista *show* redireccionar al *post* del artista, en la vista *category* se podrá visualizar las categorías de los *posts* y por último la vista *tag* muestra las etiquetas, similar a la vista categorías, en la **Figura 3.29** muestra las vistas *Blade*.

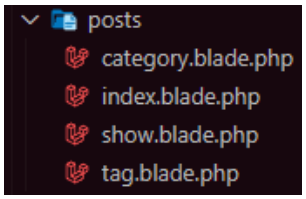


Figura 3.29: Vistas *Blade*.

Para los estilos de estas vistas se usa *Tailwind* y *Livewire*, en las etiquetas de *HTML* se usa prefijos propios de *Tailwind,* que permite tener interfaces atractivas y también el *responsive* para que se adapte las interfaces en diferentes pantallas.

Para ejemplificar como se usa las etiquetas y prefijos de *Tailwind,* la **Figura 3.30** muestra el uso de etiquetas *HTML* para formar las estructura, la etiqueta **x-app-layout** sirve para integrar componentes de *Livewire*, prefijos como **lg:px-8**, **sm:px-6** para adaptar a pantallas grades y pequeñas respetivamente.

```
<x-app-layout>
    <div class="mx-auto max-w-5xl px-4 sm:px-6 lg:px-8 py-8">
        {{--    todo mayuscula: uppercase
                tamano de la latra: text-3xL --}}
        <h1 class="uppercase text-center text-3xl font-bold text-gray-600 mb-8">
            Etiqueta: {{$tag->name}}
        </h1>

        {{-- intera la coleccion post --}}
        @foreach ($posts as $post)
            <x-card-post :post="$post">

            </x-card-post>
        @endforeach

        {{-- paginacion --}}
        <div class="mt-4">
            {{$posts->links()}}
        </div>
    </div>
</x-app-layout>
```

Figura 3.30: *Tailwind* y *Livewire*.

En la **Figura 3.31** y **Figura 3.32** muestra un ejemplo como se logra llamar a los datos desde *MySQL* en las plantillas *Blade*, en este caso en la **Figura 3.31** se

obserta el controlador ***PostController*** con un metodo *index*, donde se almacena en la variable *posts* el modelo *Post* que recueperar los *posts* de los usuarios.

```php
public function index(){
        $posts = Post::where('status', 2)->latest('id')->paginate(30); // paginate(50): numero de posts que se visualiza

        return view('posts.index', compact('posts'));
    }
```

Figura 3.31: Método *index*.

En la plantilla ***index.blade*** se dónde se llama a la variable *posts* y mediante un *foreach* itera toda la colección *posts* de los usuarios, se puede visualizar en la **Figura 3.32**.

```blade
<x-app-layout>
    <div class="container py-8">
        <div class="mx-auto max-w-7xl px-2 sm:px-6 lg:px-8
                    grid lg:grid-cols-5 md:grid-cols-3 sm:grid-cols-1 gap-6" >
            @foreach ($posts as $post)
                <a href="{{route('posts.show', $post)}}"
                    class="w-full h-80 bg-cover bg-center rounded-lg"
                    style=" background-image: url(@if($post->image) {{Storage::url($post->image->url)}} @else 1280.jpg @endif)">
                </a>
            @endforeach
        </div>

        {{-- link de paginacion --}}
        <div class="mx-auto max-w-7xl px-2 sm:px-6 lg:px-8 mt-4">
            {{$posts->links()}}
        </div>
    </div>
</x-app-layout>
```

Figura 3.32: Plantilla *index.blade*.

En la **Figura 3.33** se visualiza las rutas que se encargar de direccionar a las plantillas *Blade* como: *index, show, tag* y *category*. Para administrar las rutas se debe ir a la ruta **routes\web.php** del *framework* para realizar las respetivas configuraciones.

```php
Route::get('/', [PostController::class, 'index'])->name('posts.index');

Route::get('posts/{post}', [PostController::class, 'show'])->name('posts.show');

Route::get('category/{category}', [PostController::class, 'category'])->name('posts.category');

Route::get('tag/{tag}', [PostController::class, 'tag'])->name('posts.tag');
```

Figura 3.33: Rutas.

En la **Figura 3.34** muestra un ejemplo como se visualiza los posts de la vista *index*.

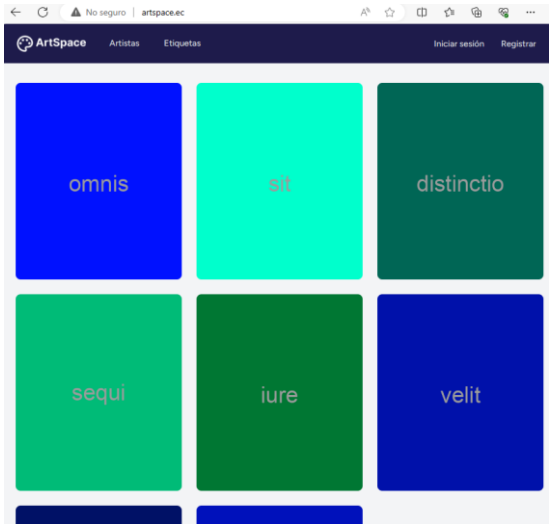


Figura 3.34: Vista *index*.

Instalación de *AdminLTE 3* para el uso de la plantilla panel de administrador

Se crea un archivo *PHP* (*admin*) en la carpeta *routes* (**routes\admin.php**), aquí se define las rutas del administrador. Para que *Laravel* reconozca el archivo (*admin*) que se creó como un archivo de ruta, se especifica en **app\Providers\RouteServiceProvider.php** y se agrega el *path* del archivo *admin*, como muestra en la **Figura 3.35**.

```php
public function boot(): void
    {
        RateLimiter::for('api', function (Request $request) {
            return Limit::perMinute(60)->by($request->user()?->id ?: $request->ip());
        });

        $this->routes(function () {
            Route::middleware('api')
                ->prefix('api')
                ->group(base_path('routes/api.php'));

            Route::middleware('web')
                ->group(base_path('routes/web.php'));

            Route::middleware('web', 'auth')
                ->prefix('admin')
                ->group(base_path('routes/admin.php'));
        });
    }
```

Figura 3.35: *RouteServiceProvider*.

Se crea el controlador *HomeController* usando el comando **C23**, **TABLA X** para poder renderizar la vista. También se crea una carpeta llamada *admin* en **resources\views** del *framework* y un archivo *Blade* llamado *index.admin*, como presenta la **Figura 3.36**.

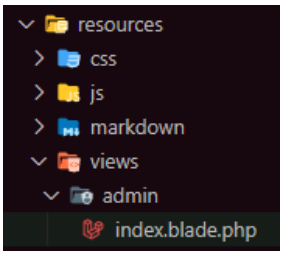


Figura 3.36: Instalación del panel de control *AdminLTE 3*.

Se define la ruta en el archivo *admin* (**routes\admin.php**) como muestra en la **Figura 3.37**.

```php
Route::get('', [HomeController::class, 'index'])->name('admin.home');
```

Figura 3.37: Ruta index admin.

Con los comandos **C24** y **C25, TABLA X** se ejecuta en la terminal de *Visual Studio Code* y se instala *AdminLTE 3,* en la **Figura 3.38** se observa el panel de

administrador que se usa en el proyecto. Es importante mencionar que se utiliza componentes de *Bootstrap* para crear estilos en el paquete de *AdminLTE 3*.

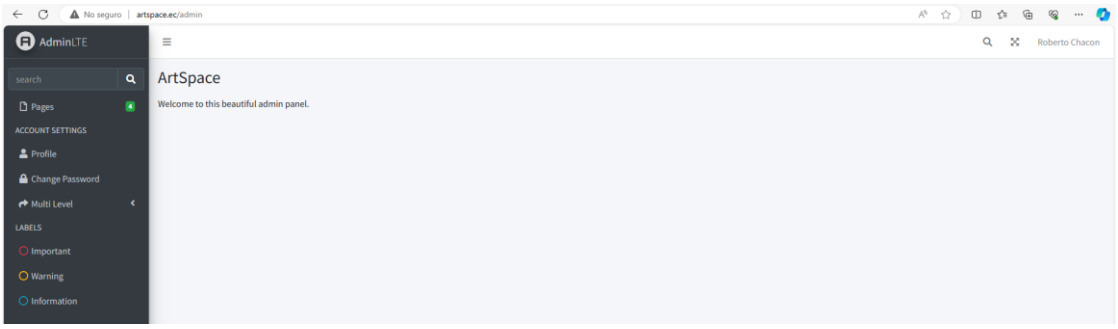


Figura 3.38: *AdminLTE 3*.

Cuando se trabaja con la plantilla *AdminLTE 3* se debe usar estilos de *Bootstrap* y para usar estilos de *Livewire* se cambia la configuración de *AdminLTE 3* en **config\adminlte.php** del *framework*. En el apartado *Livewire* por defecto está en *false*, se cambia a *true* como presenta la **Figura 3.39**.

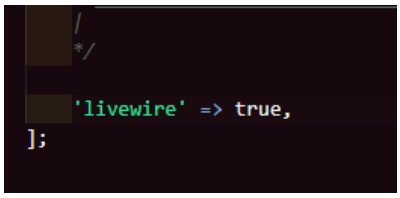


Figura 3.39: Configuración *AdminLTE 3*.

CRUD's de categorías, etiquetas y *posts*

En la ruta **app\Http\Controllers** del *framework* se crea una carpeta llamada *Admin*, en la **Figura 3.40** se observa la carpeta que contiene los controladores de categorías, etiquetas y *posts* dentro de la carpeta *Admin.*

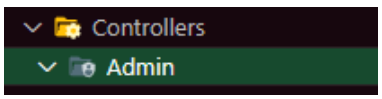


Figura 3.40: Carpeta *Admin*.

Para poder crear los CRUD's de las categorías, etiquetas y *posts* se debe crear rutas de tipo *resource*, en la **Figura 3.41** muestra las rutas.

```php
// Rutas para crear el CRUD categorias (genera un controlador con los 7 metodos del CRUD)
Route::resource('categories', CategoryController::class)->names('admin.categories');

// Rutas para crear el CRUD etiquetas (genera un controlador con los 7 metodos del CRUD)
Route::resource('tags', TagController::class)->names('admin.tags');

// Rutas para crear el CRUD posts (genera un controlador con los 7 metodos del CRUD)
Route::resource('posts', PostController::class)->names('admin.posts');
```

Figura 3.41: Rutas *resource*.

En la termina de *Visual Studio Code* se ingresa el comando **C29**, **TABLA X**, este comando permite verificar si se generó las siete rutas *resource*, en la **Figura 3.42** muestra un ejemplo de las rutas cuando se creo la ruta tipo *resource* de categorías.

```
GET|HEAD   admin/categories ..................... admin.categories.index › Admin\CategoryController@index
POST       admin/categories ..................... admin.categories.store › Admin\CategoryController@store
GET|HEAD   admin/categories/create ............... admin.categories.create › Admin\CategoryController@create
GET|HEAD   admin/categories/{category} ............... admin.categories.show › Admin\CategoryController@show
PUT|PATCH  admin/categories/{category} ........... admin.categories.update › Admin\CategoryController@update
DELETE     admin/categories/{category} ......... admin.categories.destroy › Admin\CategoryController@destroy
GET|HEAD   admin/categories/{category}/edit ......... admin.categories.edit › Admin\CategoryController@edit
```

Figura 3.42: Ruta *resource Categories*.

También se crea controladores en la ruta **app\Http\Controllers** del *framework* con los comandos **C26**, **C27** y **C28**, **TABLA X**, en la **Figura 3.43** muestra los controladores**.**

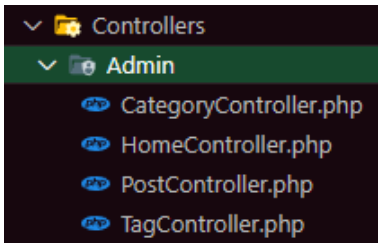
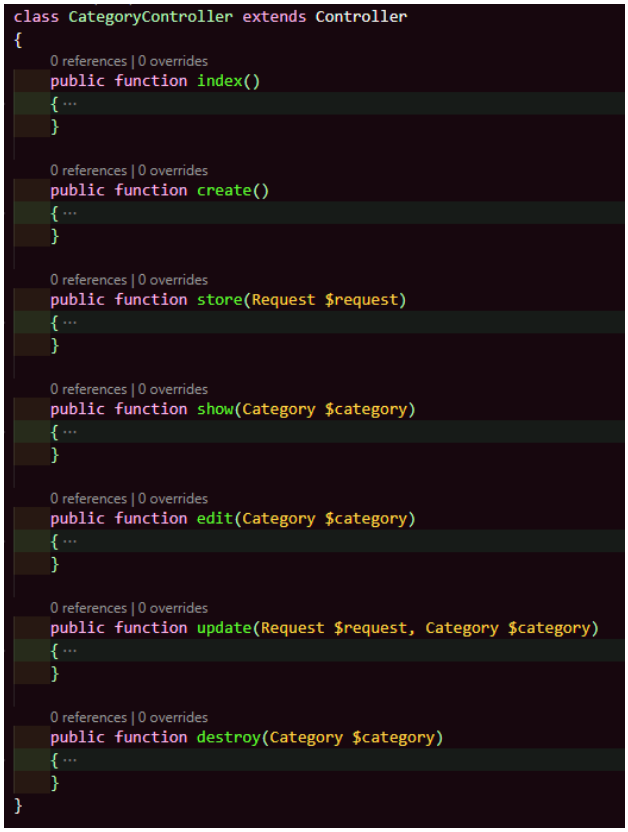


Figura 3.43: Controladores.

Los controladores *CategoryController*, *PostController* y *TagController* tienen los siete métodos del CRUD**,** en la **Figura 3.44** presenta un ejemplo del controlador *CategoryController*.



```
class CategoryController extends Controller
{
    0 references | 0 overrides
    public function index()
    { ···
    }

    0 references | 0 overrides
    public function create()
    { ···
    }

    0 references | 0 overrides
    public function store(Request $request)
    { ···
    }

    0 references | 0 overrides
    public function show(Category $category)
    { ···
    }

    0 references | 0 overrides
    public function edit(Category $category)
    { ···
    }

    0 references | 0 overrides
    public function update(Request $request, Category $category)
    { ···
    }

    0 references | 0 overrides
    public function destroy(Category $category)
    { ···
    }
}
```

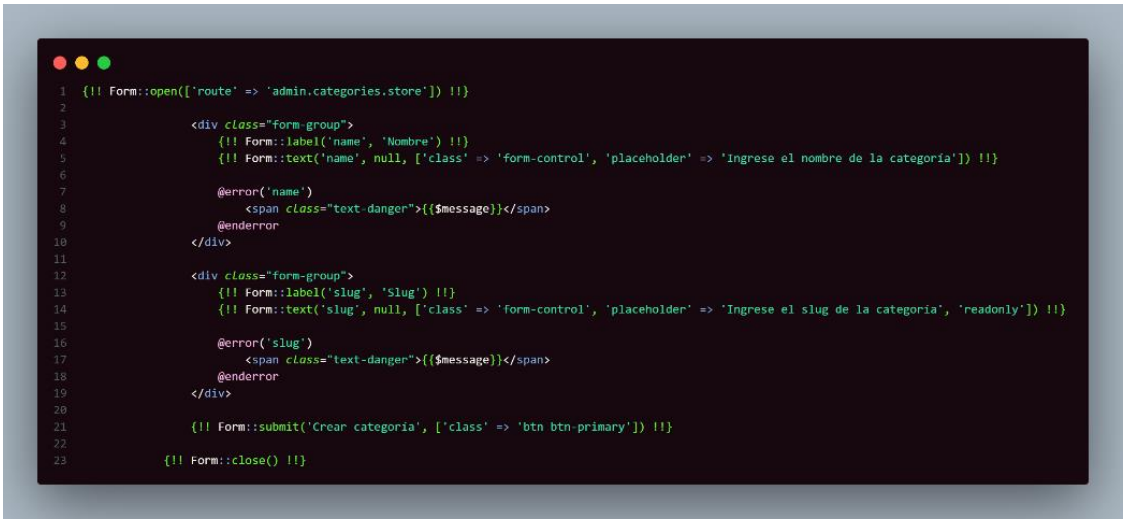
Figura 3.44: *CategoryController*.

El método *index* muestra el listado de categorías, *store* para guardar una nueva categoría, *create* para mostrar el formulario y crear una nueva categoría, *update* para actualizar una categoría y *delete* para borrar una categoría.

Visualización de formularios y validaciones

Para el uso de formularios se instala *Laravel collective* en el proyecto, se usa la última versión (v6.x) y en la terminal de *Visual Studio Code* se ingresa el comando **C30**, **TABLA X**. *Laravel collective* permite utilizar una sintaxis basada en *PHP*, hace que el código del proyecto sea más limpio y fácil de mantener.

Cuando se usa *Laravel collective* no se utiliza el token *csrf* que usualmente se usa en los formularios, así también facilita en recolectar la información de las relaciones de los modelos y colocarlos en los *inputs*, evitando escribir código en exceso al momento de recuperar información. En la **Figura 3.45** muestra un ejemplo del formulario de categorías usando *Laravel collective.*



```
{!! Form::open(['route' => 'admin.categories.store']) !!}

                <div class="form-group">
                    {!! Form::label('name', 'Nombre') !!}
                    {!! Form::text('name', null, ['class' => 'form-control', 'placeholder' => 'Ingrese el nombre de la categoría']) !!}

                    @error('name')
                        <span class="text-danger">{{$message}}</span>
                    @enderror
                </div>

                <div class="form-group">
                    {!! Form::label('slug', 'Slug') !!}
                    {!! Form::text('slug', null, ['class' => 'form-control', 'placeholder' => 'Ingrese el slug de la categoría', 'readonly']) !!}

                    @error('slug')
                        <span class="text-danger">{{$message}}</span>
                    @enderror
                </div>

                {!! Form::submit('Crear categoria', ['class' => 'btn btn-primary']) !!}

            {!! Form::close() !!}
```

Figura 3.45: Formulario de categorías con *Laravel collective.*

Para que los enlaces sean amigables se implemento un campo *slug* al momento de crear una categoría, para que el *slug* funcione se debe descargar el *plugin* de *jQuery* y colocar en la ruta **public\vendor** del *framework.*

Con un componente de *Livewire* se agregar un buscador en los formularios, se usa el comando **C31**, **TABLA X**, este buscador facilitara en encontrar rápidamente un *post* en un listado, como presenta en la **Figura 3.46**.

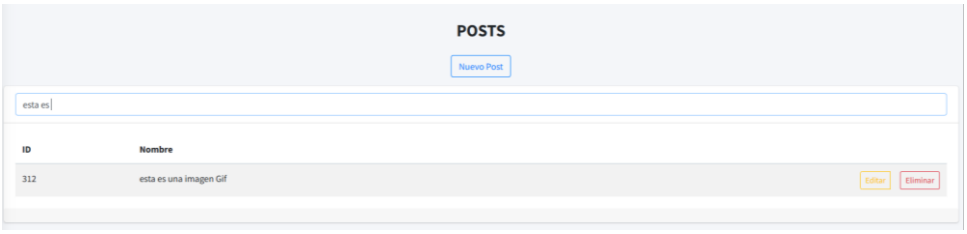


Figura 3.46: Componente *Livewire* buscador.

En las validaciones de los formularios se crea un controlador con el comando **C32**, **TABLA X** que genera un archivo en la ruta **app\Http\Requests\PostRequest.php** del *framework,* este controlador llamado *StorePostRequest* ayuda en establecer reglas de validaciones en los formularios de los *posts*, en la **Figura 3.47** muestra las reglas que se estableció en el controlador, por ejemplo el campo *name* (nombre del post), es requerido para poder crear un *post.* También es estableció una regla para que todos los campos sean requeridos si el *post* se elige la opción de estado publicado.

```php
public function rules(): array
    {
        $post = $this->route()->parameter('post');

        $rules = [
            'name' => 'required',
            'slug' => 'required|unique:posts',
            'status' => 'required|in:1,2',
            //'file' => 'image',
            'file' => 'file|mimes:jpeg,png,gif,mp4,avi',
        ];

        if ($post) {
            $rules['slug'] = 'required|unique:posts,slug,' . $post->id;
        }

        if ($this->status == 2){
            $rules = array_merge($rules, [
                'category_id' => 'required',
                'tags' => 'required',
                'extract' => 'required',
                'body' => 'required',
            ]);
        }
        return $rules;
    }
```

Figura 3.47: *StorePostRequest*.

Para que las validaciones estén en español se debe usar paquete de *Laravel Lang*, se ingresa el comando **C33**, **TABLA X** en la terminal de *Visual Studio Code,* después se puede ver las traducciones que se cambiaran automáticamente en la ruta **lang\es** del *framework*. Después en la ruta **config\app.php** del *framework* se cambia de ingles a español, como presenta la **Figura 3.48**.

Figura 3.48: Configuración de idioma.

Para que las configuraciones se apliquen en el proyecto se debe actualizar usando el comando **C34**, **TABLA X**, en la **Figura 3.49** presenta las validaciones al español.

Figura 3.49: Validaciones en español.

Un *observer* es una clase que permite agrupar eventos relacionados a un determinado modelo, dichos eventos se ejecutan cuando se realice una acción relacionado al modelo como la creación, actualización o eliminación de un registro.

Con la ayuda de los *observer* se puede eliminar una imagen cuando se elimine un post, para eliminar un post con su imagen se debe ejecutar el comando **C35**, **TABLA X**. El nombre del *observer* es *PostObserver* y se le asocia al modelo *Post* para que cuando un usuario elimine un *post* también se elimine la imagen o video relacionado con el *post*, así se evita en almacenar archivos basura como imágenes o videos.

Para registrar el *observer* que se creó, se debe ir a la ruta **app\Providers\EventServiceProvider.php** y en el método *boot* se registra el *PostObserver,* como muestra en la **Figura 3.50**.

Figura 3.50: *PostObserver*.

Reglas de autorización

Para evitar que los usuarios no alteren los *posts* de otros usuarios se utiliza los *policies* de *Laravel*, con el uso de *policies* se evita que los usuarios escriban el *id*

del *post* en la ruta y accedan al *post* de otros usuarios, así mismo en los *posts* e estado de borrador. Con el comando **C36**, **TABLA X** se genera el controlador *PostPolicy*. En la ruta app\Policies\PostPolicy.php del *framework* se crea dos métodos *autor* y *published* de autorización, como muestra en la **Figura 3.51**.

```php
// regla de autorizacion para verificar si un usuario es el autor de un post
    public function author(User $user, Post $post) {
        if ($user->id == $post->user_id) {
            return true;
        } else {
            return false;
        }
    }

    // regla de autorizacion para evitar acceso en post en estado borrador
    public function published(?User $user, Post $post) {
        if ($post->status == 2) {
            return true;
        } else {
            return false;
        }
    }
```

Figura 3.51: *PostPolicy.*

El método *autor* es una regla de autorización para verificar si un usuario es el autor de un *post*. El método *published* es una regla de autorización para evitar que los usuarios accedan a un *post* en estado borrador.

En la ruta app\Http\Controllers\Admin\PostController.php del *framework* existen métodos como: *edit*, *update* y *destroy*, en la **Figura 3.52** muestra un ejemplo del método *destroy* y como se aplica la regla se autorización para evitar que cierto usuario no elimine un *post*.

```php
public function destroy(Post $post)
    {
        // regla de autorizacion
        $this->authorize('author', $post);

        $post->delete();

        return redirect()->route('admin.posts.index', $post)->with('info', 'El post se eliminó con exitó.');
    }
```

Figura 3.52: Reglas de autorización *PostController admin.*

En la ruta **app\Http\Controllers\PostController.php** del *framework,* el controlador *PostController* se encarga de mostrar los *posts* en estado publicado, en este controlador se aplica la regla de autorización para evitar que los usuarios accedan a los *posts* en estado borrador mediante la ruta, como se visualiza en la **Figura 3.53**.

```php
public function show(Post $post){

        // regla de autorizacion
        $this->authorize('published', $post);

        //recuperamos todos los registros similares a este post
        $similares = Post::where('category_id', $post->category_id)
                        ->where('status', 2) // filtro (publicado)
                        ->where('id', '!=', $post->id) // recuepra todos los post sea distintos al post
                        ->latest('id') // ordena de manera descendente
                        ->take(4) // trae cuatro registros
                        ->get();

        return view('posts.show', compact('post', 'similares'));
    }
```

Figura 3.53: Reglas de autorización *PostController.*

3.4 *Sprint* 3. MÓDULO PARA ADMINISTRADOR/USUARIOS AUTENTIFICADOS.

Las tareas para el *Sprint* 3 son:

- Roles y permisos (*Laravel Permission*).
- Ocular botones y limitación de acceso de rutas.
- Sección de comentarios y buscadores.

Roles y permisos (*Laravel Permission*)

Los permisos son acciones para delimitar el acceso en distintas secciones dentro de la aplicación *web*, por ejemplo, en la aplicación tiene un apartado para crear *posts,* solo las personas autentificadas y asignadas un rol (*Blogger* o *Admin*) pueden crear *posts*, la funcionalidad de los permisos es impedir que los usuarios (no autentificados) no pueda modificar la información como: categorías, etiquetas y *posts*.

La base de datos generada por *Laravel Permission* se puede observar en la **Figura 3.54**, en total son seis tablas: *users*, *model_has_roles*, *roles*, *role_has_permissions*, *permissions*, *model_has_permissions*.

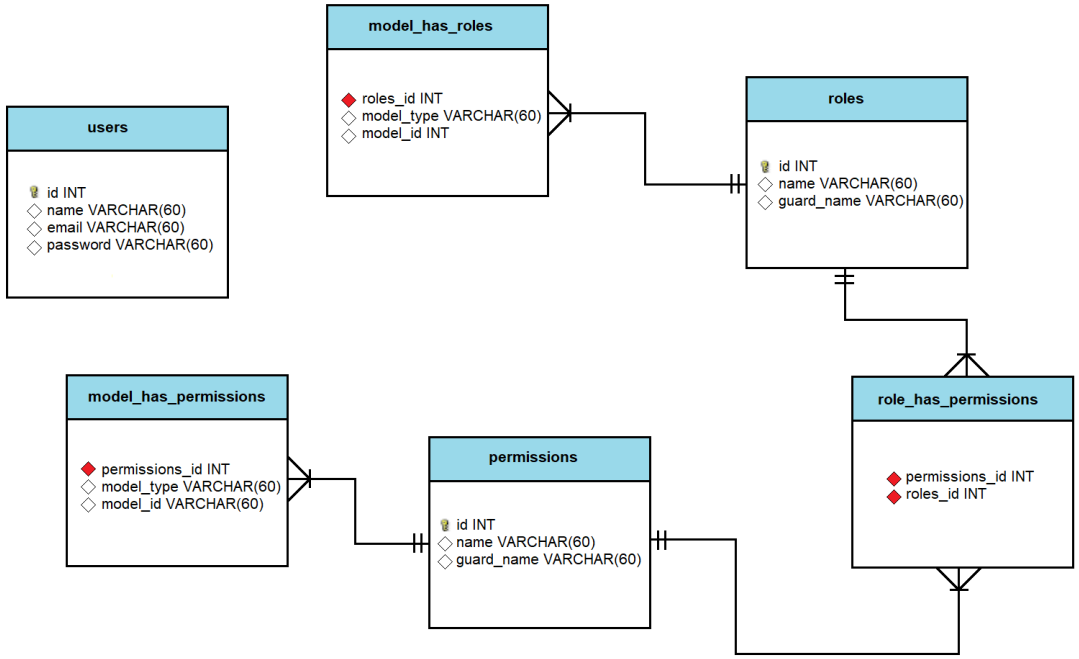


Figura 3.54: Base de datos *Laravel Permission.*

La tabla *users* se genera por defecto cuando se creó el proyecto, la tabla *model_has_roles* es una tabla intermedia, en la tabla roles se agrupan los permisos, la tabla *role_has_permissions* es una tabla intermedia, aquí se asigna los permisos a un rol, en la tabla *permissions* es donde se debe generar los permisos con código *PHP,* y la tabla *model_has_permissions* es una tabla intermedia producto de la relación de muchos a muchos de las tablas *roles* y *permissions*.

Por ejemplo, la relación que existe entre las tablas *users* – *permissions* es una relación muchos a muchos polimórfica, un usuario tendrá varios permisos y los permisos tendrán varios usuarios.

Para la instalación del paquete de *Laravel Permission* se usa el comando **C37**, **TABLA X**, los archivos descargados se alojan en la ruta **vendor\spatie\laravel-permission** del *framework,* el archivo de migración *create_permission_tables* se ubica en la ruta **vendor\spatie\laravel-**

permission\database\migrations\create_permission_tables.php.stub, este archivo se encarga de generar todas las tablas para el sistema roles y permisos.

Para usar el archivo de migración *create_permission_tables* se debe publicar con el comando **C38**, **TABLA X**. Al usar este comando hace que el archivo de migración se guarde en la ruta **database\migrations** del *framework*, sitio donde esta todas las migraciones del proyecto, en el archivo de migración (*migrations*) se puede observar en la **Figura 3.55**.

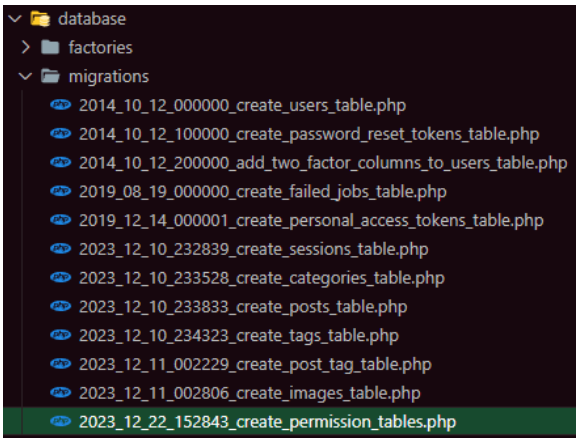


Figura 3.55: Archivo de migración.

Con el comando **C5**, **TABLA X** se ejecuta las migraciones para crear las tablas. Al usar el paquete *Laravel Permission* se crea los modelos *Permission* y *Role*, así mismo las relaciones de las tablas.

En la **Figura 3.56** muestra cómo se genera la relación del modelo *users* (**app\Models\User.php**) con los dos modelos *Permission* y *Role* (**vendor\spatie\laravel-permission\src\Models**).

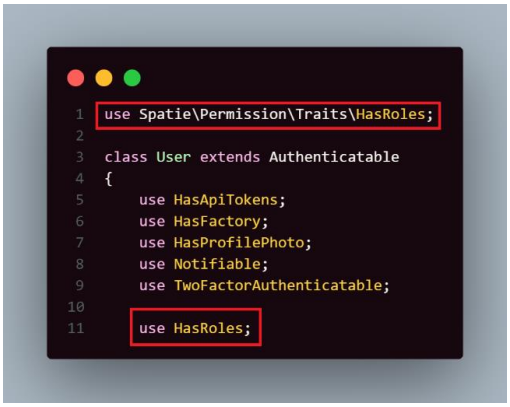


Figura 3.56: Relación del modelo *users* con los modelos *Permission* y *Role*.

Se crea un *seeder* con el comando **C39**, **TABLA X** para generar los roles *Admin* y *Blogger*, así también crear los permisos para la manipulación de información como las categorías, etiquetas y *posts*. En la **Figura 3.57** presenta un ejemplo de se asigna un rol y un permiso.

```php
public function run(): void
    {
        $role1 = Role::create(['name' => 'Admin']);
        $role2 = Role::create(['name' => 'Blogger']);

        // permiso para visualizar el dashboard
        Permission::create(['name' => 'admin.home'])->syncRoles([$role1, $role2]);

```

Figura 3.57: *Seeder RoleSedeer.*

En la ruta **database\seeders\DatabaseSeeder.php** del *framework* se dónde se llama al *seeder RoleSeeder* para que los registros de roles y permisos se almacenen en la base de datos, en la **Figura 3.58** nuestra el *seeder Roleseeder*. Con el comando **C20**, **TABLA X** se ejecuta los *seeders*.

```php
public function run(): void
    {
        // Para que se borren las imagenes cada vez que ejecute el comando "php artisan migrate:fresh --seed"
        Storage::deleteDirectory('posts'); // elimna la carpeta "posts"
        Storage::makeDirectory('posts'); // Crea la carpeta posts dentro de la carpeta storage

        // para que los registros de RoleSeeder se almacenen en la base de datos
        $this->call(RoleSeeder::class);

        $this->call(UserSeeder::class);
        Category::factory(7)->create(); // crea 7 categorias
        Tag::factory(15)->create(); // crea 15 etiquetas

        // Se descarga nuevamente las imagenes
        $this->call(PostSeeder::class); // por cada post que se cree, se crea una imagen
    }
```

Figura 3.58: *DatabaseSeeder*.

En la ruta **database\seeders\UserSeeder.php** se crea un usuario y se le asigna el rol de *Admin*, en la **Figura 3.59** muestra las credenciales del usuario administrador (*Admin*).

```php
public function run(): void
    {
        // Creo mis credenciales
        User::create([
            'name' => 'Roberto Chacón',
            'email' => 'robertochacon9514@hotmail.com',
            'password' => bcrypt('1404[redacted]'),
        ])->assignRole('Admin');

        // crear los registros falsos
        User::factory(9)->create(); // 80 registros falsos
    }
```

Figura 3.59: Asignación de roles.

En la base de datos se verifica la asignación del rol, en la **Figura 3.60** presenta la tabla *users*, la tabla intermedia *role_has_permissions* con la asignación del rol, el usuario "Roberto Chacón" tiene el rol de "*Admin"* mediante el *id* del usuario.

Tabla users:

	id	name	email
Editar Copiar Borrar	1	Roberto Chacón	robertochacon9514@hotmail.com

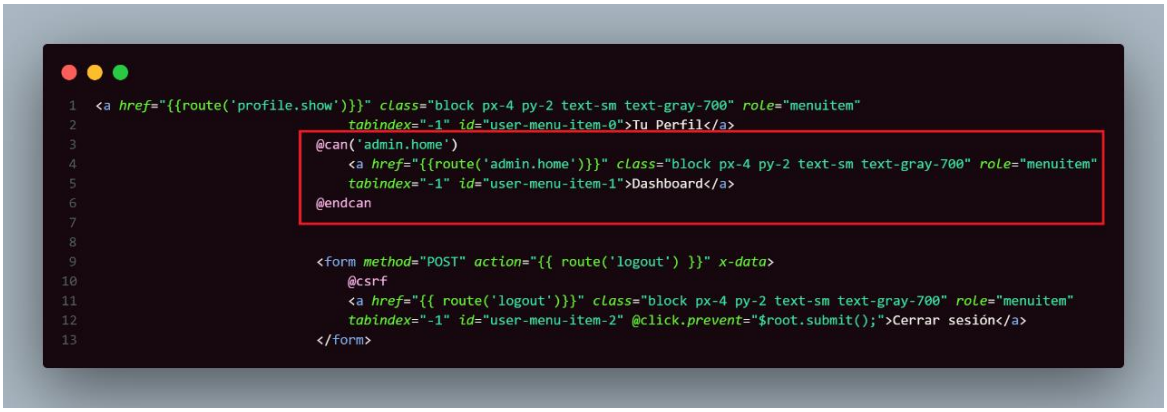
Tabla intermedia *role_has_permissions*:

	role_id	model_type	model_id
Editar Copiar Borrar	1	App\Models\User	1

Figura 3.60: Asignación de roles *MySQL.*

Ocular botones y limitación de acceso de rutas

En el menú de navegación de la página principal (*home*), se usa la directiva de *Blade*, "*can*" como muestra en la **Figura 3.61**. La directiva "*can"* verifica si el usuario tiene algún permiso, de tener el permiso para postear será visible el botón *Dashboard* y de no serlo, permanecerá oculto.



```
<a href="{{route('profile.show')}}" class="block px-4 py-2 text-sm text-gray-700" role="menuitem"
                                    tabindex="-1" id="user-menu-item-0">Tu Perfil</a>
                                @can('admin.home')
                                    <a href="{{route('admin.home')}}" class="block px-4 py-2 text-sm text-gray-700" role="menuitem"
                                    tabindex="-1" id="user-menu-item-1">Dashboard</a>
                                @endcan


                                <form method="POST" action="{{ route('logout') }}" x-data>
                                    @csrf
                                    <a href="{{ route('logout')}}" class="block px-4 py-2 text-sm text-gray-700" role="menuitem"
                                    tabindex="-1" id="user-menu-item-2" @click.prevent="$root.submit();">Cerrar sesión</a>
                                </form>
```

Figura 3.61: Ocular botones en el menú de navegación.

En el menú de navegación (*AdminLTE*) en el caso de ser un usuario que tenga el rol de administrador (*Admin*) podrá visualizar los botones de configuración como son: usuarios, categorías y etiquetas, de no ser un administrador (*Admin*) no serán visibles estos botones, en la **Figura 3.62** muestra cómo se integra la directica "*can*" de *Blade.*

```
[
            'text'        => 'Panel de control',
            'route'         => 'admin.home',
            'icon'        => 'fa fa-tachometer-alt fa-fw',
            'can'       => 'admin.home',
        ],
        [
            'text'        => 'Usuarios',
            'route'         => 'admin.users.index',
            'icon'        => 'fa fa-users fa-fw',
            'can'       => 'admin.users.index',
        ],
        [
            'text' => 'Categorias',
            'route'  => 'admin.categories.index',
            'icon' => 'fab fa-fw fa-buffer',
            'active' => ['admin/categories*'],
            'can'       => 'admin.categories.index',
        ],
        [
            'text' => 'Etiquetas',
            'route'  => 'admin.tags.index',
            'icon' => 'far fa-fw fa-bookmark',
            'active' => ['admin/tags*'],
            'can'       => 'admin.tags.index',
        ],
```

Figura 3.62: Ocular botones en el menú de navegación *AdminLTE*.

En el menú de navegación (*AdminLTE*) si un usuario que tenga el rol de *Blogger*, podrá visualizar los botones de configuración de listado de *posts* y crear un nuevo *post*, en la **Figura 3.63** muestra cómo se integra la directica "*can"* de *Blade.*

Figura 3.63: Ocular botones en el menú de navegación *AdminLTE*.

En la seguridad de acceso de rutas se pude observar en la **Figura 3.64** un ejemplo con dos usuarios, del lado izquierdo un usuario administrador y del lado derecho un usuario *Blogger*, se aprecia la diferencia en el menú de navegación.

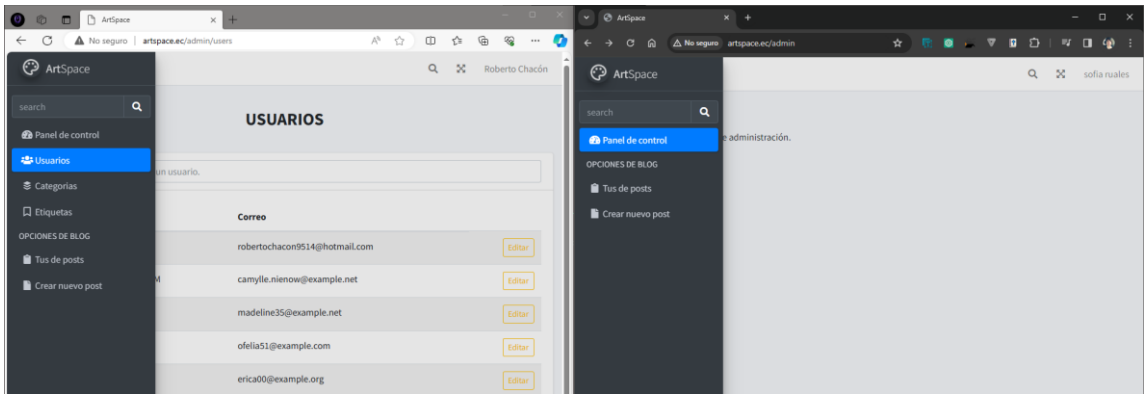


Figura 3.64: Limitación de acceso de rutas.

Para limitar el acceso de rutas, en la **Figura 3.65** muestra un ejemplo de control de seguridad de rutas, se observa como un usuario *Blogger* puede ingresar a la ruta de usuarios escribiendo en la *url* del navegador *web*.

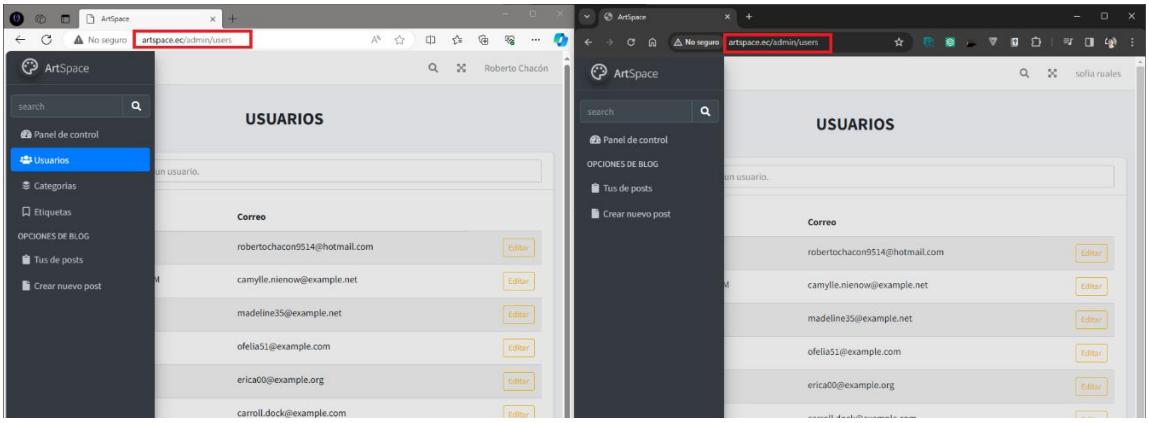


Figura 3.65: Limitación de acceso de rutas.

Para tener seguridad en el acceso de rutas se dirige a la ruta **routes\admin.php** del *framework*, en esta seccion estan las rutas del administrador, como presentra en la **Figura 3.66.**

```php
Route::get('', [HomeController::class, 'index'])->name('admin.home');

// Rutas para crear el CRUD categorias (genera un controlador con los 7 metodos del CRUD)
Route::resource('categories', CategoryController::class)->names('admin.categories');

// Rutas para crear el CRUD etiquetas (genera un controlador con los 7 metodos del CRUD)
Route::resource('tags', TagController::class)->names('admin.tags');

// Rutas para crear el CRUD posts (genera un controlador con los 7 metodos del CRUD)
Route::resource('posts', PostController::class)->names('admin.posts');

// Rutas para crear el CRUD usuarios (genera un controlador con los 7 metodos del CRUD)
Route::resource('users', UserController::class)->only(['index', 'edit', 'update'])->names('admin.users');
```

Figura 3.66: Limitación de acceso de rutas (*Admin*).

Se hace uso de *middleware* y la directiva "*can*" con el parámetro del rol "*admin.home*" para proteger esta ruta y evitar que los usuarios ingresen al panel de control del administrador (*Admin*), como se puede observar en la **Figura 3.67**.

```php
Route::get('', [HomeController::class, 'index'])->middleware('can:admin.home')->name('admin.home');
```

Figura 3.67: Limitación de acceso de rutas (*admin*).

También se debe proteger las demás rutas como las de categorías y etiquetas, en la **Figura 3.68** muestra un ejemplo como se integra un constructor en el controlador **app\Http\Controllers\Admin\UserController.php** del *framework,* para proteger el listado de usuarios y que solo sea manipulable por el administrador.

Figura 3.68: Limitación de acceso de rutas (*UserController*).

Sección de comentarios y buscadores

En la sección de comentarios se utiliza un componente de *Livewire*, se crea un componente en el cual se trabaja con un controlador que renderiza la vista en donde se refleja el funcionamiento de la sección de comentarios. Los usuarios registrados podrán comentar en la sección de comentarios. Se estableció validaciones para controlar dicha acción, en la **Figura 3.69** muestra la sección de comentarios en la vista del detalle del *post*.

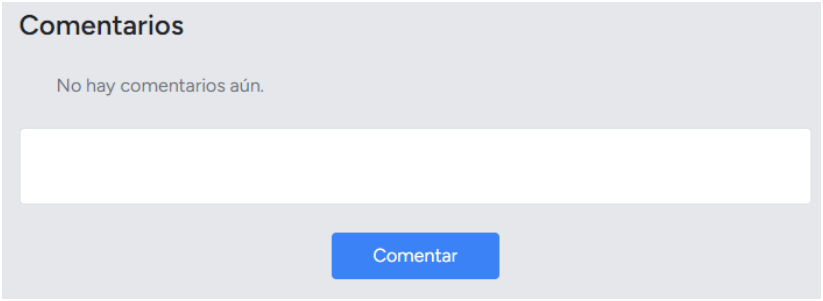


Figura 3.69: Sección de comentarios en la vista de detalle del *post*.

En la **Figura 3.70** presenta una simulación de un usuario no registrado en la aplicación *web* y como resultado se hace presente la validación.

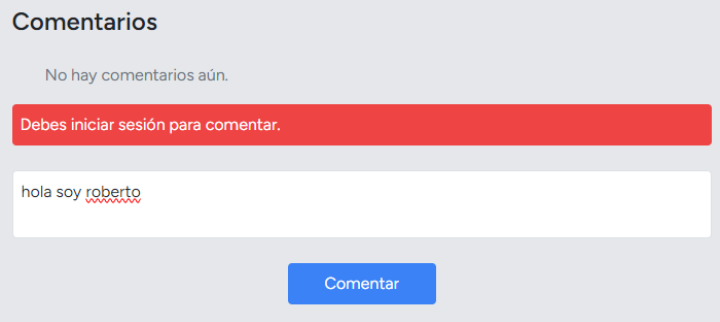


Figura 3.70: Simulación de un comentario por un usuario no registrado.

El usuario registrado en la aplicación *web*, podrá realizar comentarios en cualesquiera posts publicados por los artistas, y también podrá editar y eliminar su comentario, cabe mencionar que estas acciones serán ejecutadas por el usuario quien realizo el comentario y no otros usuarios, en la **Figura 3.71** muestra un comentario de un usuario registrado.

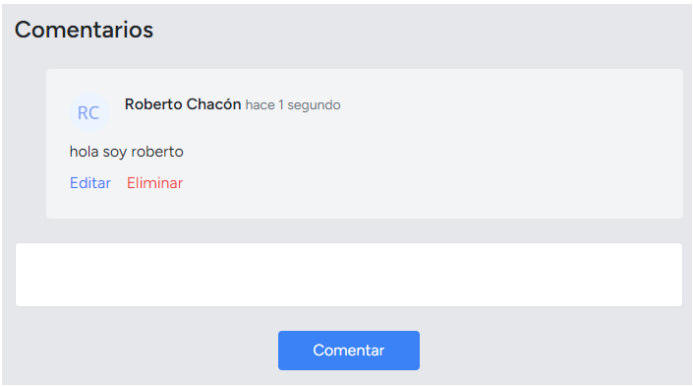


Figura 3.71: Simulación de un comentario por un usuario registrado.

La sección de comentarios de implemento con la finalidad que los usuarios alienten a los artistas en mejorar con sus obras digitales, así mismo se fomenta la autenticidad del contenido publicado.

Para que los usuarios tengan una mejor experiencia dentro de la aplicación *web* se implementó buscadores con el propósito de encontrar contenido multimedia de acuerdo con el interés del usuario, se hace uso de *Livewire* para crear los buscadores, en la **Figura 3.72** muestra el buscador de etiquetas que está en la página principal (*home*), también se implementa buscadores tanto en las vistas de los usuarios para poder buscar artistas, etiquetas y categoría, en la **Figura 3.73** muestra un ejemplo del buscador de artistas. También se implementa buscadores

en las vistas del *Dashboard,* en la **Figura 3.74** muestra el buscador en la sección de la gestión de usuarios.

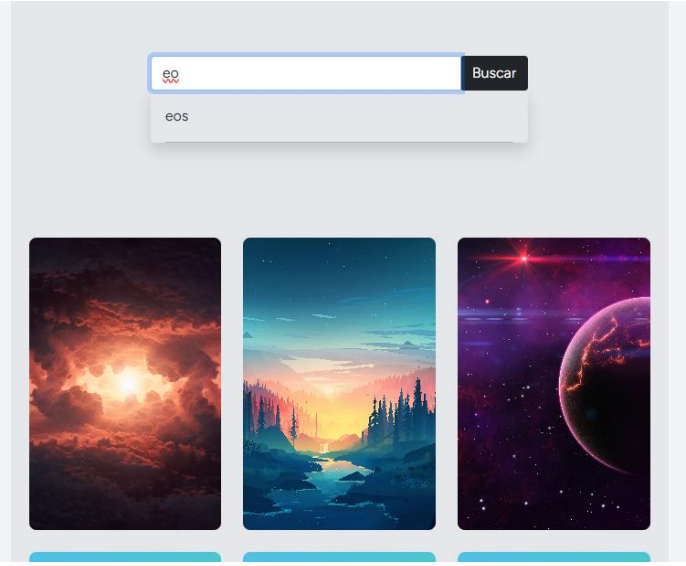


Figura 3.72: Buscador de etiquetas página principal (*home*).

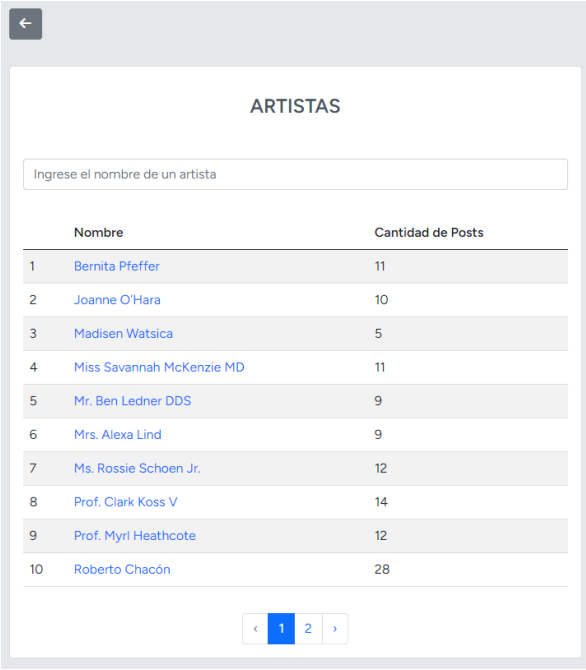


Figura 3.73: Buscador de artistas.

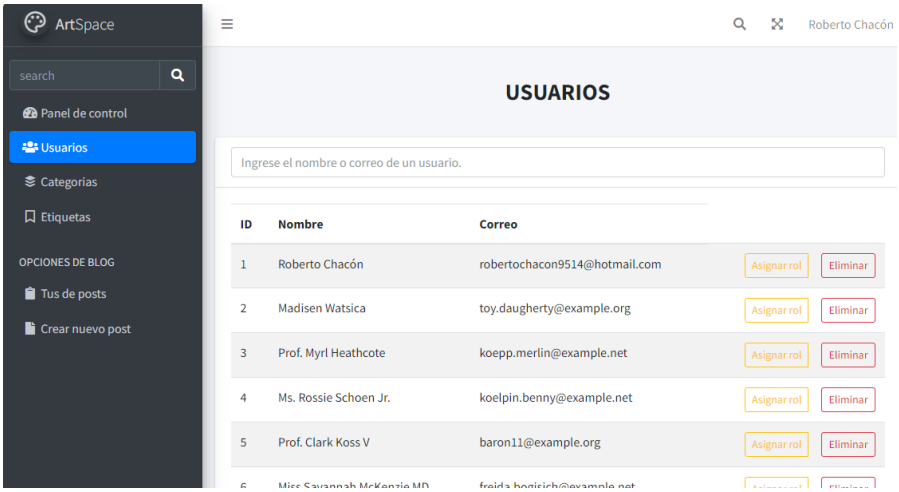


Figura 3.74: Buscador de usuarios en *Dashboard*.

3.5 *Sprint* 4. PRUEBAS

Las tareas para el *Sprint* 4 son:

- Pruebas de compatibilidad.
- Pruebas de rendimiento.
- Pruebas de aceptación.

Pruebas de compatibilidad

Se utiliza la herramienta *Browserstack* para realizar las pruebas de compatibilidad en la aplicación *web*, se realizó pruebas de compatibilidad en los navegadores más usados en el mundo como *Google*, *Mozilla Firefox* y *Opera*. En la **Figura 3.75** presenta el panel de control donde permite elegir en distintos navegadores.

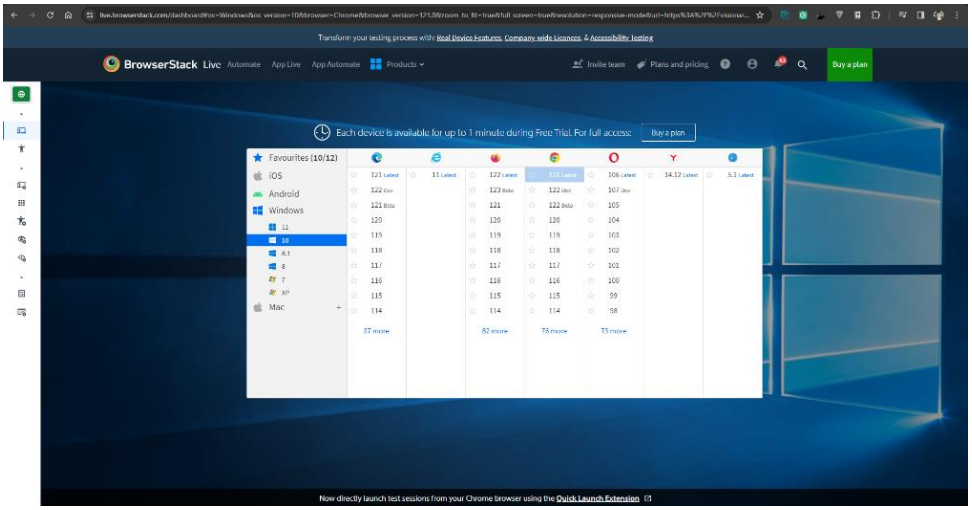


Figura 3.75: Panel de control *Browserstack*.

En la **Figura 3.76** se observa el resultado de compatibilidad con el navegador *Google*.

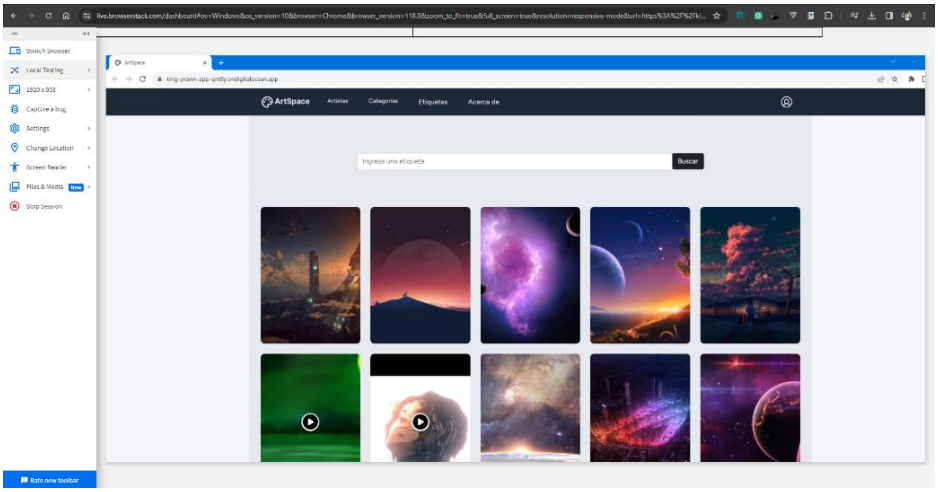


Figura 3.76: Compatibilidad con el navegador *Google*.

En la **Figura 3.77** se observa el resultado de compatibilidad con el navegador *Mozilla Firefox.*

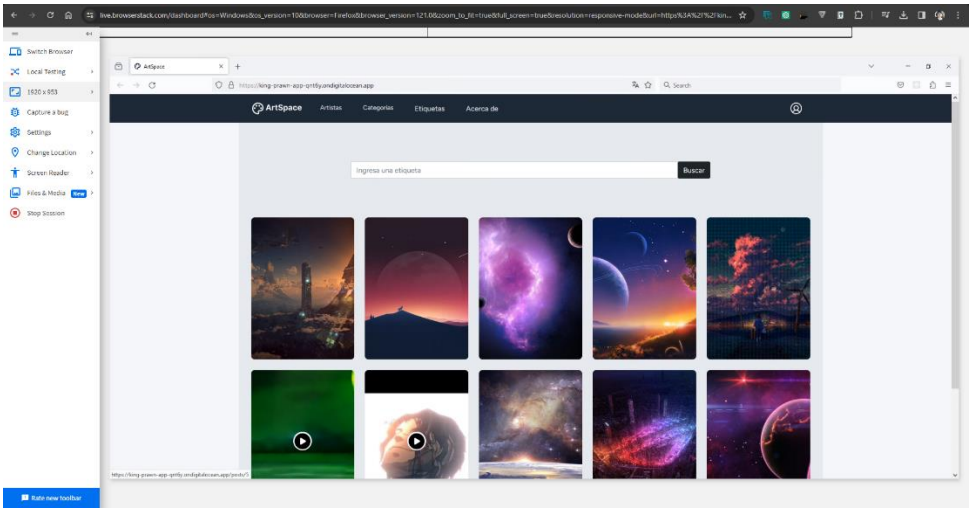


Figura 3.77: Compatibilidad con el navegador *Google*.

En la **Figura 3.78** se observa el resultado de compatibilidad con el navegador *Opera.*

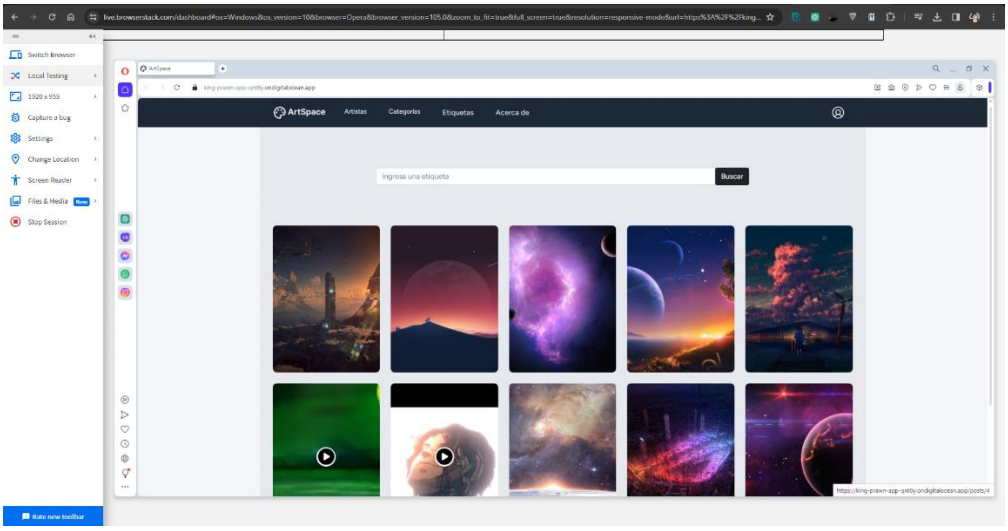


Figura 3.78: Compatibilidad con el navegador *Opera*.

Pruebas de rendimiento

Para saber el rendimiento de la aplicación *web*, se usa la herramienta *PageSpeed Insights* proporcionada por *Google* para evaluar y analizar el rendimiento en términos de velocidad y optimización.

Para usar la herramienta *PageSpeed Insights* se debe ingresar la *url* de la aplicación *web*, posteriormente se tiene una puntuación de rendimiento y recomendaciones para mejorar la velocidad de carga y la optimización de la página.

También se obtiene la accesibilidad tanto para dispositivos móviles como para ordenador.

En la **Figura 3.79** presenta el informe en detalle sobre la aplicación *web*, se obtiene un rendimiento del 87%, accesibilidad del 95% y *SEO* de 91%.

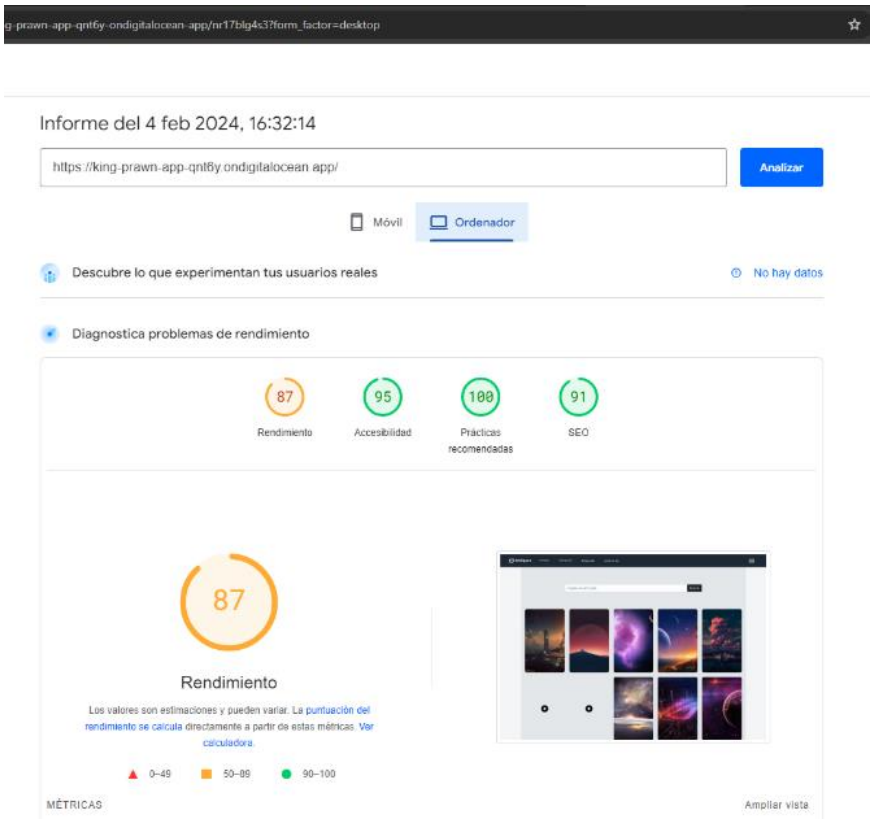


Figura 3.79: Pruebas de rendimiento para *web* en *PageSpeed Insights*.

También *PageSpeed Insights* ofrece un diagnóstico para mejorar aspectos importantes en la aplicación *web*, en la **Figura 3.80** se puede observar como en la aplicación *web* no tiene contenido *CSS*, en el desarrollo se usó *Bootstrap* y *Tailwind* para el maquetado de las interfaces.

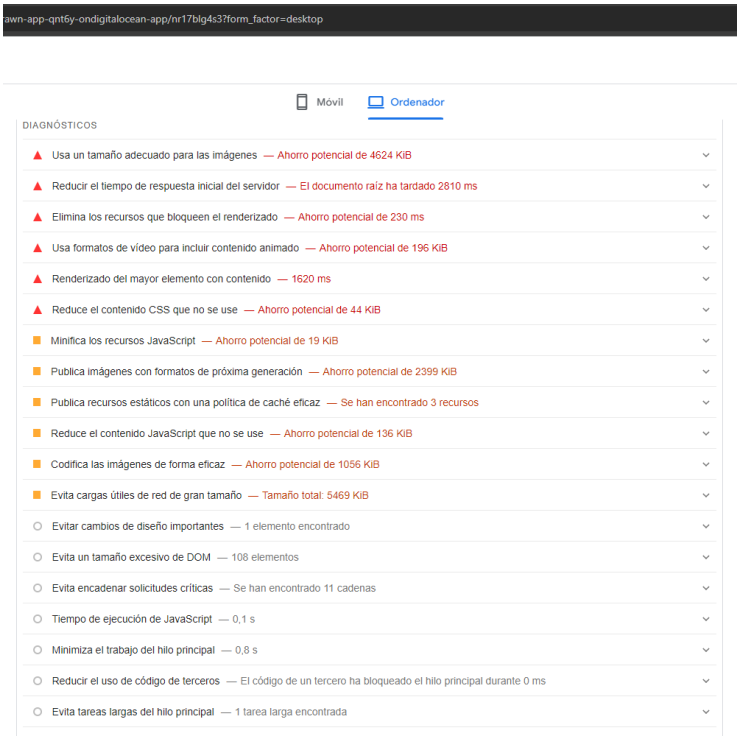


Figura 3.80: Diagnóstico de *PageSpeed Insights*.

Se realizó una prueba de rendimiento para dispositivos móviles, en la **Figura 3.81** presenta el informe en detalle y se obtiene un rendimiento del 64%, accesibilidad del 95% y *SEO* de 92%.

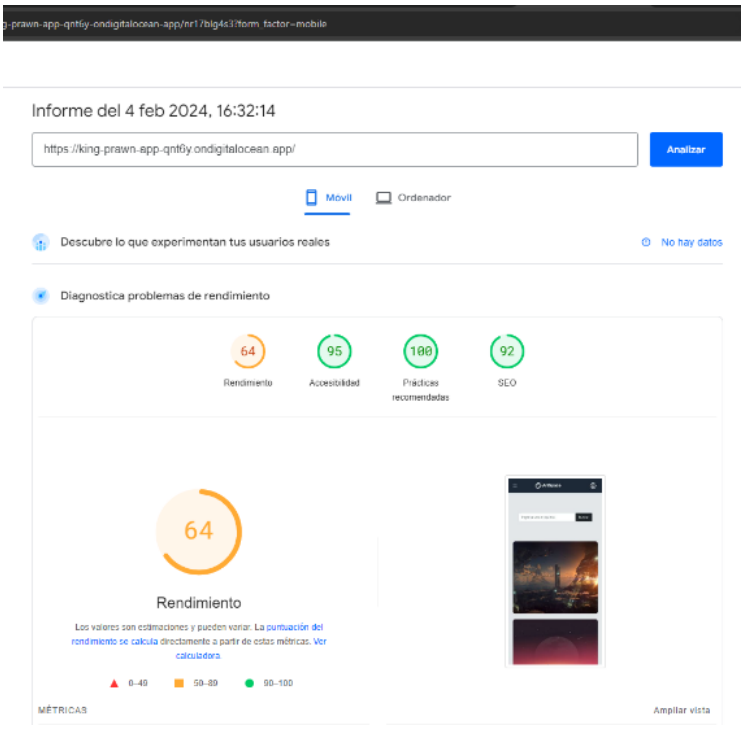


Figura 3.81: Pruebas de rendimiento para móviles en *PageSpeed Insights*.

Pruebas de aceptación

Las pruebas de aceptación se encuentran en el **ANEXO II**, **Pruebas de aceptación**. Un ejemplo se presenta en la **TABLA V**, contienen una descripción sobre la prueba de aceptación realizada con un usuario que está familiarizado en sitios *web* sobre para subir contenido multimedia.

TABLA V: Prueba de aceptación – MÓDULO PARA ADMINISTRADOR/USUARIOS AUTENTIFICADOS.

PRUEBA DE ACEPTACIÓN	
Identificador: PA006	**Historia de Usuario:** HU006
Nombre: Gestión de roles y mantenimiento de datos	
Descripción: El usuario administrador establece el tipo de rol y permisos, así mismo eliminar los roles asignados. También necesita modificar y eliminar datos, para que la aplicación web tenga mejor rendimiento con la información	
Procedimientos de ejecución: • Para iniciar sesión como administrador (*Admin*) dar clic en el icono "*users*" en la página principal, parte superior derecho: o Despliegue el menú de opciones del icono "users" y clic en "iniciar sesión". o En el formulario de "*login*" llenar los campos correo y contraseña. o Dar clic en el botón "iniciar sesión". • Para ingresar al panel de control (*Dashboard*): o Ingresar al panel de control (*Dashboard*) en el menú de navegación de la página principal, clic en el icono "*users*" y seleccionar la opción "*Dashboard*" • Para asignar un rol a un usuario: o Despliegue el menú de opciones del panel de control. o Dar clic en la pestaña de "usuarios" y clic en el botón "Asignar rol". o Conceder un rol (*Blogger* o *Admin*) a un usuario. • Para eliminar datos como categorías, etiquetas, usuarios o *post* de usuarios: o Despliegue el menú de opciones del panel de control. o Dar clic en las pestañas de "usuarios", "*Post* de usuarios", "Categorías" y "Etiquetas".	

○ Eliminar registros con el botón “Eliminar”.
El resultado esperado es: Se permite a un usuario autentificado con rol de *Admin* asignar un rol a los usuarios autentificados.
Evaluación: Se evalúa el resultado deseado y el usuario lo aprueba completamente.

3.6 *Sprint* 5. DESPLIEGUE

Las tareas para el *Sprint* 5 son:

- Desplegar la aplicación *web* en el servidor *Digital Ocean*.

Desplegar la aplicación *web* en el servidor *Digital Ocean*.

Una vez terminado el desarrollo de la aplicación *web* se debe alojar en internet para que los internautas puedan conocer el funcionamiento y el objetivo por el cual se creó esta aplicación *web*, que es de brindar espacios virtuales a los artistas digitales para que comercialicen sus productos.

En este caso se usa el servidor de *Digital Ocean* para poder hacer *deploy* de la aplicación *web*. *Digital Ocean* ofrece una máquina virtual para poder hacer configuraciones personalizadas, tanto en el *deploy* como en producción

Para que la aplicación *web* este en producción, es necesario crear una cuenta de *GitHub* y subir el proyecto a un repositorio, en la **Figura 3.82** muestra vinculación el repositorio de *GitHub* con *Digital Ocean*.

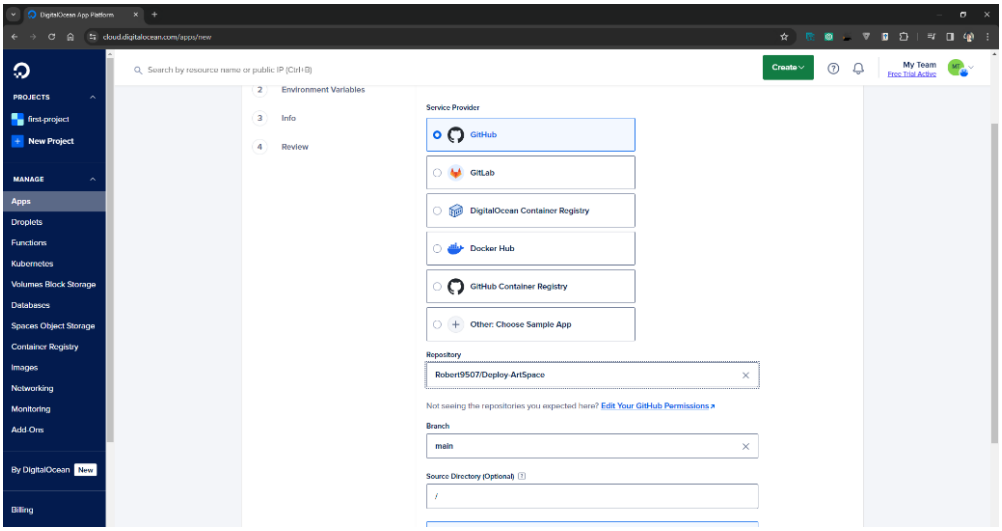


Figura 3.82: Panel de control de *Digital Ocean*.

Digital Ocean carga el *framework* de *Laravel* a su servidor y cada vez se modifica en el código fuente (repositorio de *GitHub*), *Digital Ocean* realizara los cambios de manera automática. También ofrece una consola para poder realizar las migraciones de las tablas, como presenta a **Figura 3.83**.

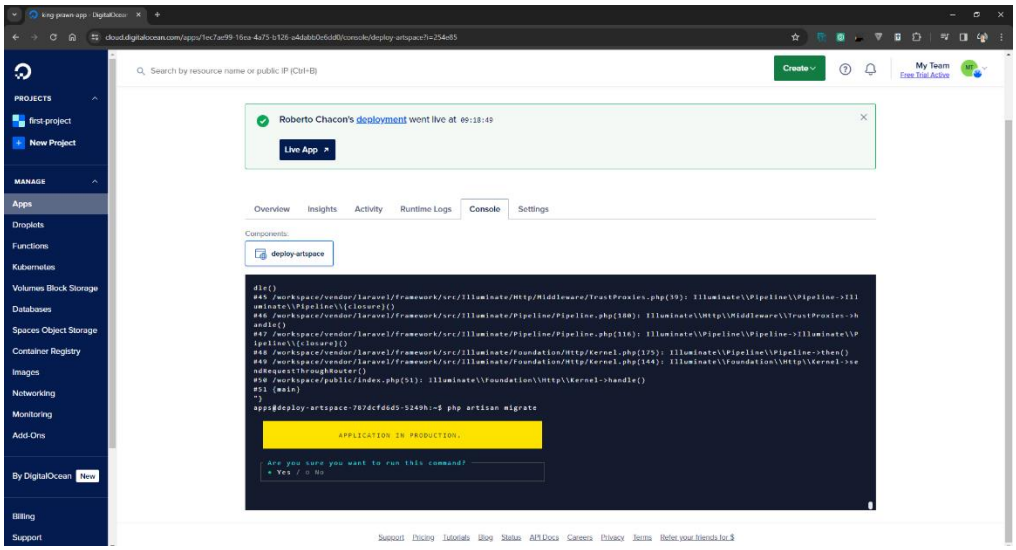


Figura 3.83: *Deploy* del *framework Laravel* en *Digital Ocean*.

4 CONCLUSIONES

En el estudio para el trabajo de integración curricular (TIC) se puede concluir que:

- La aplicación *web* cumple todos los requerimientos basados en las necesidades para la comunidad artística digital. Tiene interfaces intuitivas para que el usuario pueda tener buena experiencia en la navegación de las diferentes vistas que ofrece la aplicación *web*. También se implementó buscadores para filtrar contenido multimedia y que el usuario pueda encontrar contenido de su interés más rápido.

- La arquitectura MVC (modelo-vista-controlador) de *Laravel* permite modularizar, mantenimiento y la escalabilidad del código. Gracias a la arquitectura MVC se puede integrar más módulos y funcionalidades en la aplicación *web* en un futuro, con la finalidad de que la aplicación sea más productiva.

- La aplicación *web* “ArtSpace” cumple con el objetivo principal de proporcionar espacios comerciales para que los artistas digitales puedan exhibir y comercializar sus productos hacia los internautas, mediante interfaces intuitivas y fáciles de comprender para que los usuarios puedan navegar en la aplicación *web* sin dificultad.

- Las pruebas de rendimiento muestran un buen desempeño al optimizar los recursos y la velocidad de carga en la aplicación web. También se tiene una alta optimización de búsqueda, es decir, los motores de búsquedas de los navegadores tienen más resultados relevantes sobre la aplicación *web* y por consecuencia atraiga mas visitantes.

5 RECOMENDACIONES

Al finalizar el desarrollo del proyecto de realiza algunas recomendaciones para que el proyecto funcione de manera eficiente:

- Durante la etapa de desarrollo se debe realizar una profunda indagación sobre las herramientas y tecnologías como *framework*, lenguaje de programación, entre otros, para que sean compatibles entre sí y no provoque un retraso en la entrega del producto final.
- No se recomienda usar repositorios de terceros *(plugins*) porque puede causar un daño a nivel de código, como incoherencia en la gestión de datos, al momento de realizar una petición al servidor o falta de lógica en el sistema. Los repositorios de terceros en algunos casos son versiones antiguas y puede no ser compatible con el entorno de desarrollo que se esté usando en ese momento, tal motivo, se recomienda usar (*plugins*) de fuentes oficiales y actualizadas.
- Antes de realizar el *deploy* del proyecto, se recomienda investigar el servidor en donde se alojará la aplicación *web*, existen servidores que ofrecen todas las configuraciones ya echas, para que el desarrollador solamente se preocupe en el código fuente. En el caso de *Digital Ocean* se debe configurar desde cero, como los puertos que se conectan a la base de datos, las migraciones de las tablas, entre otras configuraciones más. Si no se tiene mucha experiencia en las configuraciones de algún servidor, se recomienda el uso de servidores con configuraciones ya establecidas, por ejemplo*, Laravel Forge*.

6 REFERENCIAS BIBLIOGRAFÍCAS

[1] B. V. Ridge, «Medium Multimedia,» Abril 2023. [En línea]. Available: https://www.mediummultimedia.com/diseno/como-es-el-arte-digital-en-la-actualidad/. [Último acceso: 20 Diciembre 2023].

[2] I. N. d. P. Industrial, «INAPI,» 2024. [En línea]. Available: https://www.inapi.cl/portal/institucional/600/w3-article-839.html. [Último acceso: 20 Diciembre 2023].

[3] H. Sheykin, «FINMODELSLAB,» 19 Agosto 2023. [En línea]. Available: https://finmodelslab.com/es/blogs/sell-business/how-to-sell-online-digital-art-store. [Último acceso: 20 Diciembre 2023].

[4] J. Cárdenas, «rockcontent,» 26 Febrero 2022. [En línea]. Available: https://rockcontent.com/es/blog/politica-de-privacidad/. [Último acceso: 22 Diciembre 2023].

[5] 16 Marzo 2021. [En línea]. Available: https://www.ibm.com/docs/es/radfws/9.6.1?topic=modeling-physical-data-models. [Último acceso: 23 Diciembre 2023].

[6] B. V. Ridge, «Medium Multimedia,» 7 Octubre 2023. [En línea]. Available: https://www.mediummultimedia.com/diseno/cual-es-la-diferencia-entre-diseno-grafico-y-diseno-grafico/. [Último acceso: 23 Diciembre 2023].

[7] P. Canal, «iebs,» 4 Mayo 2023. [En línea]. Available: https://www.iebschool.com/blog/diseno-centrado-en-el-usuario-analitica-usabilidad/#:~:text=El%20dise%C3%B1o%20centrado%20en%20el%20usuario%20(DCU)%20es%20una%20metodolog%C3%ADa,la%20conceptualizaci%C3%B3n%20hasta%20la%20implementaci%C3%B3n.. [Último acceso: 23 Diciembre 2023].

[8] B. V. Ridge, «Medium Multimedia,» 7 Octubre 2023. [En línea]. Available: https://www.mediummultimedia.com/diseno/cual-es-el-objetivo-de-la-carrera-de-diseno-grafico/#:~:text=El%20dise%C3%B1o%20gr%C3%A1fico%20es%20una%20disciplina%20creativa%20que%20se%20enfoca,manera%20clara%2C%20efectiva%20y%20est%C3%A9tica.. [Último acceso: 23 Diciembre 2023].

[9] N. E. A. C. Leiva, «Linkedin,» 28 Febrero 2023. [En línea]. Available: https://www.linkedin.com/pulse/icebergs-de-la-accesibilidad-cap%C3%ADtulo-9-el-impacto-del-cabeza-leiva/?originalSubdomain=es. [Último acceso: 23 Diciembre 2023].

[10] «Cursosdesarrolloweb,» 19 Enero 2023. [En línea]. Available: https://www.cursosdesarrolloweb.es/blog/modelos-en-laravel-

8#:~:text=En%20Laravel%2C%20los%20modelos%20son,insertar%2C%20actualizar%20y%20eliminar%20datos.. [Último acceso: 23 Diciembre 2023].

[11] V. A. Flores, «Coders Free,» [En línea]. Available: https://codersfree.com/courses-status/aprende-laravel-desde-cero/vistas-en-laravel. [Último acceso: 23 Diciembre 2023].

[12] V. A. Flores, «Coders Free,» [En línea]. Available: https://codersfree.com/courses-status/aprende-laravel-desde-cero/controladores-en-laravel-que-son-y-para-que-sirven. [Último acceso: 23 Diciembre 2023].

[13] C. Cosmin, «Kinsta,» 3 Enero 2023. [En línea]. Available: https://kinsta.com/es/blog/rutas-laravel/. [Último acceso: 23 Diciembre 2023].

[14] «Gitbooks,» [En línea]. Available: https://richos.gitbooks.io/laravel-5/content/capitulos/chapter10.html?q=. [Último acceso: 23 Diciembre 2023].

[15] «Laravel,» [En línea]. Available: https://laravel.com/docs/5.0/eloquent. [Último acceso: 23 Diciembre 2023].

[16] «Laravel,» [En línea]. Available: https://laravel.com/docs/5.0/migrations#main-content. [Último acceso: 23 Diciembre 2023].

[17] V. G. Adán, «Linkedin,» 12 Septiembre 2017. [En línea]. Available: https://www.linkedin.com/pulse/pruebas-de-compatibilidad-en-diferentes-navegadores-con-g%C3%B3mez-ad%C3%A1n/?originalSubdomain=es. [Último acceso: 23 Diciembre 2023].

[18] «IBM,» 6 Marzo 2021. [En línea]. Available: https://www.ibm.com/docs/es/rtw/9.0.0?topic=phases-performance-testing. [Último acceso: 23 Diciembre 2023].

[19] L. A. Training, «Linkedin,» 11 Abril 2020. [En línea]. Available: https://www.linkedin.com/pulse/que-son-las-pruebas-de-aceptaci%C3%B3n-los-andes-training/?originalSubdomain=es. [Último acceso: 23 Diciembre 2023].

[20] A. Zorro, «crehana,» 22 Abril 2022. [En línea]. Available: https://www.crehana.com/blog/transformacion-digital/digital-ocean/. [Último acceso: 23 Diciembre 2023].

[21] F. F. Perera, «epsilon eridani,» [En línea]. Available: http://www.epsilon-eridani.com/cubic/ap/cubic.php/doc/phpDocumentor---documentacion-para-codigo-PHP-246.html. [Último acceso: 23 Diciembre 2023].

[22] «cloudflare,» [En línea]. Available: https://www.cloudflare.com/es-es/learning/privacy/what-is-data-

privacy/#:~:text=La%20privacidad%20de%20los%20datos%20es%20la%20protecci%C3%B3n%20de%20los,acceder%20a%20su%20informaci%C3%B3n%20personal.. [Último acceso: 23 Diciembre 2023].

[23] «PowerData,» [En línea]. Available: https://www.powerdata.es/integracion-de-datos. [Último acceso: 23 Diciembre 2023].

[24] V. A. Flores, «Coders Free,» 6 Agosto 2023. [En línea]. Available: https://codersfree.com/posts/los-beneficios-de-utilizar-un-framework-como-laravel-en-el-desarrollo-web. [Último acceso: 23 Diciembre 2023].

[25] «Powerdesigner,» 2023. [En línea]. Available: https://www.powerdesigner.biz/ES/powerdesigner/powerdesigner-features.html. [Último acceso: 23 Diciembre 2023].

[26] «KEEPCODING,» [En línea]. Available: https://keepcoding.io/blog/herramientas-de-ux-y-ui-mas-usadas/#Framer_X. [Último acceso: 23 Diciembre 2023].

[27] P. Londoño, «HubSpot,» 17 Enero 2023. [En línea]. Available: https://blog.hubspot.es/website/que-es-mysql. [Último acceso: 23 Diciembre 2023].

[28] «Kinsta,» 19 Junio 2023. [En línea]. Available: https://kinsta.com/es/base-de-conocimiento/instalar-laravel/. [Último acceso: 23 Diciembre 2023].

[29] F. Flores, «OpenWebinars,» 22 Julio 2022. [En línea]. Available: https://openwebinars.net/blog/que-es-visual-studio-code-y-que-ventajas-ofrece/. [Último acceso: 23 Diciembre 2023].

[30] Jesús, «DONGEE,» 25 Abril 2022. [En línea]. Available: https://www.dongee.com/tutoriales/que-es-xampp/. [Último acceso: 23 Diciembre 2023].

[31] M. Clemente, «rockcontent,» 22 Abril 2020. [En línea]. Available: https://rockcontent.com/es/blog/digital-ocean/. [Último acceso: 23 Diciembre 2023].

[32] «Desarrollo de Software,» [En línea]. Available: https://desarrollodesoftware.dev/metodologia/. [Último acceso: 24 Diciembre 2023].

[33] «Deloitte,» [En línea]. Available: https://www2.deloitte.com/es/es/pages/technology/articles/roles-y-responsabilidades-scrum.html. [Último acceso: 24 Diciembre 2023].

[34] «miro,» [En línea]. Available: https://miro.com/es/agile/que-son-artefactos-scrum/. [Último acceso: 24 Diciembre 2023].

[35] «asana,» 13 Noviembre 2022. [En línea]. Available: https://asana.com/es/resources/requirements-gathering. [Último acceso: 24 Diciembre 2023].

[36] M. REHKOPF, «ATLASSIAN,» [En línea]. Available: https://www.atlassian.com/es/agile/project-management/user-stories. [Último acceso: 26 Diciembre 2023].

[37] «Deloitte,» [En línea]. Available: https://www2.deloitte.com/es/es/pages/technology/articles/artefactos-scrum.html. [Último acceso: 26 Diciembre 2023].

[38] «seoestudios,» 21 Enero 2021. [En línea]. Available: https://www.seoestudios.es/que-es-un-mockup/. [Último acceso: 26 Diciembre 2023].

[39] C. V. H. Otálora, «KONRAD LORENZ,» 7 Mayo 2018. [En línea]. Available: https://repositorio.konradlorenz.edu.co/handle/001/138. [Último acceso: 26 Diciembre 2023].