

MANUAL TECNICO DEL SISTEMA



MERCARQ

Contenido

<i>Contenido</i>	2
<i>Lista de Figuras</i>	4
<i>Lista De Tablas</i>	5
<i>Presentación</i>	c
<i>Resumen</i>	7
<i>Objetivo</i>	8
<i>Finalidad Del Manual</i>	8
<i>Introducción</i>	5
1. Aspectos Técnicos	10
1.1. Herramientas Utilizadas Para El Desarrollo	10
1.1.1. Visual Studio Code	10
1.1.2. GitHub.....	10
1.1.3. Laravel	10
1.1.4. PhpMyAdmin.....	11
1.1.5. MySQL.....	11
1.1.6. Bootstrap 5.....	11
1.1.7. GitBash.....	11
1.1.8. Librerías Utilizadas en Laravel	11
2.3. Diccionario de Datos	13
3. Aspecto Técnico Del Desarrollo Del Sistema	1c
3.1. Modificación Local.....	16
<i>Explicación Código</i>	25
Controladores	25
BlueprintController	25
HomeController	25
InvitationController.....	26
ProfileController.....	27
PurchaseController.....	27
Solicitudes - Request.....	28
LoginRequest.....	28
ProfileUpdateRequest.....	29
FreeDownloadOwnerNotification	29

WhatsAppPurchaseNotification	30
Modelos - Models	30
Blueprint.....	30
Purchase	31
Solicitud	31
User	32
Migraciones- Migrations.....	32
Migración Crear Usuarios.....	32
Migración create_cache.....	33
Migración Create_Jobs.....	33
Migración create_blueprints.....	34
Migración Purchase.....	35
Migración add_avatar_path.....	35
Migración add_whatsapp_Blueprints.....	36
Migración create_solicitudes.....	36
Migración add_missing_columns.....	37
MANUALES	38
Vistas-Views	39
RUTAS - Routes	39
Guardado de documentos.....	40
<i>4. Requerimientos Del Software.....</i>	<i>42</i>
<i>Bibliografía</i>	<i>43</i>

Lista de Figuras

<i>Figura 1. Página web de descarga de PHP</i>	<i>1c</i>
<i>Figura 2 Página de descarga de Visual Studio Code</i>	<i>17</i>
<i>Figura 3 Página de descarga de Laragon</i>	<i>17</i>
<i>Figura 4 Página de descarga de Composer</i>	<i>18</i>
<i>Figura 5 Clonacion repositorio Github</i>	<i>15</i>
<i>Figura c Visualización del Proyecto En VSC</i>	<i>15</i>
<i>Figura 7 Carpetas utilizadas y Patrón MVC</i>	<i>20</i>
<i>Figura 8 Cambios repositorio GitHub.....</i>	<i>22</i>
<i>Figura S Validación Composer.Json</i>	<i>23</i>
<i>Figura 10 inicialización en servidor local</i>	<i>23</i>
<i>Figura 11 Ingreso a la administración de PhpMyAdmin</i>	<i>24</i>
<i>Figura 12 Administración en PhpMyAdmin</i>	<i>24</i>
<i>Figura 13 BlueprintController.....</i>	<i>25</i>
<i>Figura 14 HomeController</i>	<i>25</i>
<i>Figura 15 InvitationController</i>	<i>2c</i>
<i>Figura 1c ProfileController.....</i>	<i>27</i>
<i>Figura 17 PurchaseController.....</i>	<i>27</i>
<i>Figura 18 LoginRequest</i>	<i>28</i>
<i>Figura 1S ProfileUpdateRequest</i>	<i>25</i>
<i>Figura 20 FreeDownloadOwnerNotification.....</i>	<i>25</i>
<i>Figura 21 WhatsAppPurchaseNotification</i>	<i>30</i>
<i>Figura 22 Modelo Blueprint</i>	<i>30</i>
<i>Figura 23 Modelo Purchase</i>	<i>31</i>
<i>Figura 24 Modelo Solicitud</i>	<i>31</i>
<i>Figura 25 Modelo User</i>	<i>32</i>
<i>Figura 2c Migración crear Usuarios.....</i>	<i>32</i>
<i>Figura 27 Migración create_cache</i>	<i>33</i>

<i>Figura 28 Migración Create_Jobs</i>	<i>33</i>
<i>Figura 2S Migración create_blueprints</i>	<i>34</i>
<i>Figura 30 Migración Purchase</i>	<i>35</i>
<i>Figura 31 add_avatar_path</i>	<i>35</i>
<i>Figura 32 add_whatsapp_Blueprints</i>	<i>3c</i>
<i>Figura 33 create_solitudes</i>	<i>37</i>
<i>Figura 34 add_missing_columns.....</i>	<i>38</i>
<i>Figura 35 Manuales del Sistema</i>	<i>3S</i>
<i>Figura 3c Vistas.....</i>	<i>3S</i>
<i>Figura 37 Routes</i>	<i>40</i>
<i>Figura 38 Almacenamiento de Documentos en la plataforma.....</i>	<i>41</i>

Lista De Tablas

<i>Diccionario de datos modelo solicitudes.....</i>	<i>13</i>
<i>Diccionario de datos modelo purchases</i>	<i>13</i>
<i>Diccionario de datos modelo blueprints</i>	<i>14</i>
<i>Diccionario de datos modelo users.....</i>	<i>15</i>

Presentación

Este manual contiene toda la información necesaria para utilizar, instalar y mantener el software de gestión de planos arquitectónicos Mercarq. El sistema está diseñado para facilitar la administración y distribución de planos de arquitectura, permitiendo obtener un control preciso sobre cada plano disponible para la venta. Además, proporciona una visión clara del flujo y comportamiento de las cotizaciones de clientes, así como de las transacciones realizadas y otros procesos relacionados.

La gestión eficaz del catálogo de planos arquitectónicos es esencial para mantener las operaciones comerciales y optimizar los procesos de venta y distribución. Ehrhardt y Brigham (2007) afirman que la gestión de inventarios digitales tiene tres objetivos principales:

- Garantizar la disponibilidad continua de planos para los clientes
- Mantener un catálogo organizado y actualizado para facilitar la búsqueda y selección
- Optimizar los procesos de venta para maximizar la rentabilidad del negocio

Manejar grandes catálogos de planos arquitectónicos puede resultar complicado y costoso, especialmente con técnicas tradicionales como la "gestión manual de archivos", que requiere recursos humanos significativos y tiempo considerable, siendo además propensa a errores de clasificación y pérdida de información. Esta situación es común en empresas arquitectónicas y de diseño que manejan múltiples proyectos y variaciones de planos.

Desde hace tiempo, la venta segura de planos arquitectónicos ha enfrentado importantes dificultades en la gestión del catálogo digital, procesamiento de pagos seguros, protección de la propiedad intelectual y entrega eficiente de archivos, debido a métodos de gestión ineficaces y plataformas inadecuadas para este tipo de productos especializados.

En respuesta a esta necesidad, este manual detalla el diseño y funcionamiento de la plataforma web Mercarq, que permite a los usuarios registrarse, explorar el catálogo de planos arquitectónicos y realizar pagos seguros por medio de WhatsApp, con escalabilidad futura para implementar pasarelas de pago integradas. La plataforma conecta arquitectos y diseñadores con clientes potenciales, mostrando planos de manera atractiva para promocionarlos efectivamente. El sistema administra de forma eficiente toda la información sobre los planos comercializados, incluyendo precios, especificaciones técnicas y estado de disponibilidad. Además, el manual incluye una guía detallada para desarrolladores que deseen conocer, utilizar o realizar mejoras en el software, proporcionando una comprensión profunda de la estructura y arquitectura de desarrollo del aplicativo.

Resumen

La revolución digital ha transformado radicalmente la comercialización de productos arquitectónicos, exigiendo soluciones innovadoras que trasciendan los métodos tradicionales de gestión. En este contexto emerge Mercarq, una plataforma web vanguardista que redefine los paradigmas establecidos en la venta de planos arquitectónicos, integrando tecnología de punta con una experiencia de usuario excepcional que satisface las demandas del mercado contemporáneo.

El ecosistema Mercarq constituye una arquitectura digital robusta que amalgama funcionalidades de registro intuitivo, catálogo dinámico y sistema de pagos híbrido a través de WhatsApp, estableciendo un puente tecnológico entre arquitectos visionarios y clientes exigentes. Esta plataforma trasciende las limitaciones convencionales del comercio electrónico especializado, ofreciendo una gestión integral que optimiza cada eslabón de la cadena de valor desde la catalogación hasta la entrega final.

La propuesta de valor se fundamenta en tres pilares estratégicos: la democratización del acceso a planos arquitectónicos de calidad premium, la protección integral de la propiedad intelectual mediante sistemas de seguridad avanzados, y la creación de un Marketplace especializado que fomenta la conexión directa entre creadores y consumidores. Esta sinergia genera un entorno comercial dinámico donde la innovación arquitectónica encuentra su canal de expresión más efectivo.

Este manual representa la culminación de un proceso de software meticulosamente diseñado, proporcionando una guía comprehensiva que abarca desde la perspectiva del usuario final hasta los aspectos más técnicos del desarrollo. La documentación establece un estándar de excelencia que facilita la adopción, mantenimiento y evolución continua de la plataforma, asegurando su sostenibilidad y crecimiento en el competitivo mercado de soluciones arquitectónicas digitales.

Objetivo

Proporcionar una guía integral y estructurada que facilite la comprensión, implementación y utilización óptima de la plataforma web Mercarq, habilitando a usuarios, administradores y desarrolladores para aprovechar al máximo las capacidades del sistema de gestión y comercialización de planos arquitectónicos, garantizando una experiencia fluida, segura y eficiente en todos los niveles de interacción con la plataforma.

Finalidad Del Manual

Este manual tiene como finalidad proporcionar la documentación técnica completa del software Mercarq, una plataforma web para la gestión y comercialización de planos arquitectónicos. El documento está dirigido específicamente a desarrolladores, programadores y personal técnico que requieran comprender la arquitectura del sistema, realizar mantenimiento, implementar nuevas funcionalidades o modificar el código existente. La finalidad es facilitar el trabajo de desarrollo mediante la documentación detallada de la estructura del código, base de datos, APIs, procedimientos de instalación y configuración, así como las mejores prácticas para el mantenimiento y escalabilidad del software.

Introducción

El presente manual técnico tiene como objetivo principal brindar una guía completa y detallada sobre el software Mercarq desde una perspectiva técnica e informática. Su contenido está diseñado para dotar al personal encargado de la administración, edición y configuración del aplicativo con los conocimientos y herramientas necesarias para realizar estas tareas de manera apropiada y eficiente.

El documento se encuentra estructurado en las siguientes secciones:

- **Aspectos Teóricos:** En esta sección, se abordarán los conceptos fundamentales, definiciones y explicaciones teóricas relacionadas con los diferentes componentes de la plataforma web de planos arquitectónicos. Esto permitirá al lector adquirir una comprensión sólida del funcionamiento del sistema de gestión de catálogo digital y las herramientas involucradas.
- **Aspectos Técnicos Del Desarrollo Del Sistema:** Se buscará profundizar en los aspectos técnicos del desarrollo de la plataforma, incluyendo detalles sobre el almacenamiento de datos de planos arquitectónicos, la estructura del código fuente, las tecnologías web utilizadas, la integración con WhatsApp para pagos y las mejores prácticas para garantizar un uso adecuado y óptimo del aplicativo.
- **Requerimientos Del Software:** En esta sección, se detallarán los requisitos de hardware, software y configuración necesarios para el correcto funcionamiento de Mercarq. Esto permitirá a los usuarios preparar adecuadamente su entorno de trabajo y asegurar una experiencia fluida durante la implementación y el uso del sistema de gestión de planos.

A lo largo de este manual, se encontrarán instrucciones paso a paso, ejemplos prácticos y recomendaciones que facilitarán la comprensión y el dominio de Mercarq desde una perspectiva técnica.

1. Aspectos Técnicos

Mercarq es una plataforma web diseñada para facilitar la gestión eficiente de planos arquitectónicos en el mercado digital. Este sistema permite un control preciso sobre el catálogo de planos disponibles para la comercialización, brindando una visión clara del flujo y comportamiento de las transacciones, cotizaciones y procesos de venta realizados en la plataforma.

Este manual está destinado exclusivamente al personal autorizado para administrar, editar o configurar la plataforma Mercarq. Se recomienda no compartir ni divulgar la información contenida en este documento, ya que podría comprometer la integridad y seguridad de los datos almacenados en la base de datos del sistema.

1.1. Herramientas Utilizadas Para El Desarrollo

En esta sección, se detallan las herramientas informáticas empleadas para el desarrollo de la plataforma Mercarq, incluyendo lenguajes de programación, frameworks, bases de datos y entornos de desarrollo integrados (IDEs).

1.1.1. Visual Studio Code

Es un editor de código fuente ligero y potente desarrollado por Microsoft. Fue el entorno de desarrollo principal utilizado por el equipo para la edición, depuración y desarrollo del código fuente de la plataforma. Es una herramienta multiplataforma, gratuita y de código abierto que ofrece soporte nativo para JavaScript, TypeScript, Node.js y una amplia gama de extensiones que facilitan el desarrollo web. Su interfaz intuitiva, sistema de autocompletado inteligente y capacidades de debugging integradas lo convierten en una opción ideal para proyectos web modernos.

1.1.2. GitHub

Plataforma líder mundial de alojamiento de repositorios de código fuente y control de versiones basada en Git. Se utilizó un repositorio en GitHub para el proyecto Mercarq, lo que permitió al equipo de desarrollo colaborar de manera segura y coordinada. GitHub facilita el seguimiento de cambios en el código, la gestión de ramas de desarrollo, la resolución de conflictos y la implementación de flujos de trabajo colaborativos. Además, proporciona herramientas esenciales como la gestión de issues, pull requests, wikis de documentación, sistemas de integración continua y despliegue automatizado, garantizando un desarrollo organizado y profesional.

1.1.3. Laravel

Framework de desarrollo web de código abierto para PHP, reconocido por su sintaxis elegante, expresiva y fácil comprensión. Laravel fue seleccionado como la base principal para construir el backend y la lógica de negocio de la plataforma Mercarq, aprovechando su robusta arquitectura Modelo-Vista-Controlador (MVC). Este framework ofrece características avanzadas como el sistema de enrutamiento RESTful, migraciones de base de datos, ORM Eloquent para manejo de

datos, sistema de autenticación y autorización integrado, motor de plantillas Blade, y un ecosistema completo de paquetes y librerías que aceleran significativamente el desarrollo de aplicaciones web seguras y escalables.

1.1.4. PhpMyAdmin

Herramienta de código abierto escrita en PHP que proporciona una interfaz web intuitiva para administrar servidores MySQL. En el desarrollo de Mercarq, se utilizó PhpMyAdmin en conjunto con Laravel para realizar operaciones de consulta, creación, modificación y eliminación de datos en la base de datos MySQL del aplicativo de manera sencilla, facilitando la gestión del catálogo de planos y las transacciones.

1.1.5. MySQL

Sistema de gestión de bases de datos relacionales (RDBMS) de código abierto y ampliamente utilizado en el desarrollo web. MySQL se utilizó como el sistema de base de datos principal para almacenar y gestionar los datos del aplicativo Mercarq, aprovechando su integración nativa con Laravel y PHP.

1.1.6. Bootstrap 5

Framework de front-end de código abierto y ampliamente popular, que proporciona una colección de herramientas y componentes de interfaz de usuario (UI) basados en HTML, CSS y JavaScript. En Mercarq, se utilizó Bootstrap para construir una interfaz de usuario atractiva y responsive, destacando el uso de colores naranjas en el diseño, lo que permite una experiencia visualmente atractiva y adaptable a diferentes dispositivos, optimizando la promoción de planos arquitectónicos.

1.1.7. GitBash

Aplicación de línea de comandos que proporciona una emulación de la Shell de Bash en sistemas operativos Windows, lo que permite utilizar herramientas de línea de comandos de Unix. En el desarrollo de Mercarq, GitBash se utilizó en conjunto con Visual Studio Code y GitHub para realizar operaciones de control de versiones, como la creación de commits, el envío de cambios al repositorio remoto y la fusión de ramas de código.

1.1.8. Librerías Utilizadas en Laravel

El desarrollo de Mercarq incorporó diversas librerías de Laravel y paquetes adicionales para optimizar su funcionalidad. Las siguientes son las principales dependencias utilizadas:

Producción:

- **PHP:** Versión mínima requerida de PHP para ejecutar el proyecto.
- **laravel/framework:** - El núcleo del framework Laravel, que proporciona la estructura MVC y funcionalidades esenciales.

- **laravel/tinker:** - Herramienta de consola interactiva para depurar y probar código en Laravel.

Desarrollo y Pruebas:

- **fakerphp/faker:** - Generador de datos ficticios para pruebas y seeding de bases de datos.
- **laravel/breeze-** Paquete para autenticación y scaffolding de interfaces de usuario simplificadas.
- **laravel/pail:** - Herramienta de registro (logging) avanzada para monitoreo en tiempo real.
- **laravel/pint:** - Herramienta de formateo de código para mantener consistencia en el estilo del código.
- **laravel/sail-** Entorno de desarrollo con contenedores Docker para simplificar la configuración local.
- **mockery/mockery-** Biblioteca para crear mocks y pruebas unitarias.
- **nunomaduro/collision-** Mejora las excepciones con una interfaz más legible para depuración.
- **pestphp/pest:** - Framework de pruebas ligero y moderno.
- **pestphp/pest-plugin-laravel:** - Extensión de Pest para integrarse con Laravel.

Estas librerías se gestionaron mediante Composer, asegurando una instalación y actualización eficientes de las dependencias del proyecto.

2.3. Diccionario de Datos

Diccionario de datos modelo solicitudes

Nombre de la Columna	Tipo de Dato	Descripción
id	bigint UNSIGNED NOT NULL	Identificador único de la solicitud
blueprint_id	bigint UNSIGNED DEFAULT NULL	Identificador del plano relacionado
tipo_solicitud	varchar(255) NOT NULL	Tipo de solicitud (e.g., whatsapp, descarga_gratuita)
nombre_solicitante	varchar(255) NOT NULL	Nombre del solicitante
email_solicitante	varchar(255) DEFAULT NULL	Correo del solicitante
telefono_solicitante	varchar(255) DEFAULT NULL	Teléfono del solicitante
mensaje	text	Mensaje o detalles de la solicitud
ip_address	varchar(255) DEFAULT NULL	Dirección IP del solicitante
created_at	timestamp NULL DEFAULT NULL	Fecha y hora de creación
updated_at	timestamp NULL DEFAULT NULL	Fecha y hora de última actualización

Fuente: Por los autores

Diccionario de datos modelo purchases

Nombre de la Columna	Tipo de Dato	Descripción
id	bigint UNSIGNED NOT NULL	Identificador único de la compra
user_id	bigint UNSIGNED NOT NULL	Identificador del usuario comprador
blueprint_id	bigint UNSIGNED NOT NULL	Identificador del plano comprado
amount	decimal(10,2) NOT NULL	Monto de la compra
payment_method	varchar(255) DEFAULT NULL	Método de pago (e.g., WhatsApp)
payment_reference	varchar(255) DEFAULT NULL	Referencia del pago
status	varchar(255) NOT NULL DEFAULT 'completed'	Estado de la compra (por defecto 'completed')
created_at	timestamp NULL DEFAULT NULL	Fecha y hora de creación
updated_at	timestamp NULL DEFAULT NULL	Fecha y hora de última actualización

Fuente: Por los autores

Diccionario de datos modelo blueprints

Nombre de la Columna	Tipo de Dato	Descripción
id	bigint UNSIGNED NOT NULL	Identificador único del plano
user_id	bigint UNSIGNED NOT NULL	Identificador del usuario creador
title	varchar(255) NOT NULL	Título del plano
description	text	Descripción del plano
file_path	varchar(255) NOT NULL	Ruta del archivo del plano
price	int NOT NULL DEFAULT '0'	Precio del plano (en pesos)
whatsapp_number	varchar(20) NOT NULL	Número de WhatsApp para pagos
file_size	bigint DEFAULT NULL	Tamaño del archivo en bytes
is_public	tinyint(1) NOT NULL DEFAULT '0'	Indica si el plano es público (0/1)
created_at	timestamp NULL DEFAULT NULL	Fecha y hora de creación
updated_at	timestamp NULL DEFAULT NULL	Fecha y hora de última actualización

Fuente: Por los autores

Diccionario de datos modelo users

Nombre de la Columna	Tipo de Dato	Descripción
id	bigint UNSIGNED NOT NULL	Identificador único del usuario
name	varchar(255) NOT NULL	Nombre del usuario
email	varchar(255) NOT NULL	Correo electrónico del usuario (único)
avatar_path	varchar(255) DEFAULT NULL	Ruta del avatar del usuario
email_verified_at	timestamp NULL DEFAULT NULL	Fecha y hora de verificación del email
password	varchar(255) NOT NULL	Contraseña encriptada del usuario
role	varchar(255) NOT NULL DEFAULT 'cliente'	Rol del usuario (por defecto 'cliente')
remember_token	varchar(100) DEFAULT NULL	Token para recordar sesión
created_at	timestamp NULL DEFAULT NULL	Fecha y hora de creación
updated_at	timestamp NULL DEFAULT NULL	Fecha y hora de última actualización

Fuente: Por los autores

3. Aspecto Técnico Del Desarrollo Del Sistema

En la siguiente sección se procede a realizar una descripción detallada sobre los aspectos técnicos del aplicativo relacionado con la instalación de las herramientas necesarias para realizar modificaciones requeridas de manera ordenada

3.1. Creación e Integración de Docker:

Requisitos:

Tener instalado docker en el equipo, este se puede instalar directamente de la página oficial de docker: docker.com

¿Que es docker?:

Docker es una plataforma que permite crear, desplegar y ejecutar aplicaciones en contenedores. Estos contenedores son entornos ligeros, portátiles y consistentes que incluyen todo lo necesario para ejecutar una aplicación: código, dependencias, configuraciones y sistema operativo.

¿Por qué usar Docker en este proyecto?

- ✓ Ambiente unificado: Todos los miembros del equipo trabajan con la misma configuración.
- ✓ Sin conflictos de dependencias: Evita problemas entre versiones de PHP, Node, MySQL, etc.
- ✓ Fácil de levantar y derribar: Puedes iniciar o eliminar todo el entorno con un solo comando.
- ✓ Portabilidad: Se puede desplegar el mismo entorno en cualquier servidor o máquina local sin cambios.
- ✓ Aislamiento: Cada servicio (PHP, Node, MySQL) corre en su propio contenedor, sin interferencias.
- ✓ Sin descargar: Cada programa que se necesite para hacer funcionar el proyecto no es necesario descargarla en el computador ya que al levantar un contenedor instala dentro de este todo lo necesario sin necesidad de ocupar espacio o hacer mas procedimientos

Estructura en carpeta del proyecto:

```
|— Dockerfile           # Imagen personalizada para PHP y Node
|— docker-compose.yml   # Orquestador de servicios
|— .env                 # Variables de entorno para Laravel
|— mysql_custom.cnf     # Configuración personalizada de MySQL
|— package.json         # Dependencias de Vite/Node
|— composer.json        # Dependencias de Laravel/PHP
```

Pasos para crear el contenedor:

1. Se crean dos archivos:
 - a. Dockerfile: El Dockerfile define una imagen personalizada con dos etapas:
 - i. Etapa Node (Vite): Instala dependencias front-end (npm install)
 - ii. Etapa PHP-FPM (Laravel): Instala PHP 8.3, extensiones necesarias, y Composer

```
# Etapa 1: Node (solo para herramientas frontend)
FROM node:24 as nodebuilder

WORKDIR /app

COPY package*.json ./

RUN npm install

# Etapa 2: PHP + Laravel
FROM php:8.3-fpm

RUN apt-get update && apt-get install -y \
    libpng-dev \
    libonig-dev \
    libxml2-dev \
    zip \
    unzip \
    curl \
    git \
    && docker-php-ext-install pdo pdo_mysql

COPY --from=composer:latest /usr/bin/composer /usr/bin/composer

WORKDIR /var/www/html

RUN chown -R www-data:www-data /var/www/html && chmod -R 755 /var/www/html
```

- b. docker-compose.yml: Este archivo orquesta y define 4 servicios que funcionan juntos:
- app: Servidor Laravel ejecutado con php artisan server
 - db: Base de datos MySQL 8.0
 - vite: Frontend con Vite para desarrollo en caliente
 - composer: Ejecuta Composer de forma aislada

```
services:
  app:
    build: .
    ports:
      - "8000:8000"
    volumes:
      - ./var/www/html
    depends_on:
      - db
    working_dir: /var/www/html
    command: >
      sh -c "php artisan serve --host=0.0.0.0 --port=8000"

  db:
    image: mysql:8.0
    environment:
      MYSQL_DATABASE: mercado
      MYSQL_ROOT_PASSWORD: secret
    ports:
      - "3306:3306"
    volumes:
      - dbdata:/var/lib/mysql
      - ./mysql_custom.cnf:/etc/mysql/conf.d/custom.cnf

  vite:
    build:|
    context: .
    dockerfile: Dockerfile
    target: nodebuilder
    volumes:
      - ./app
    ports:
      - "5173:5173"
    working_dir: /app
    command: sh -c "npm install && npm run dev"

  composer:
    image: composer:latest
    volumes:
      - ./app
    working_dir: /app
    command: ["composer", "install"]

volumes:
  dbdata:
```

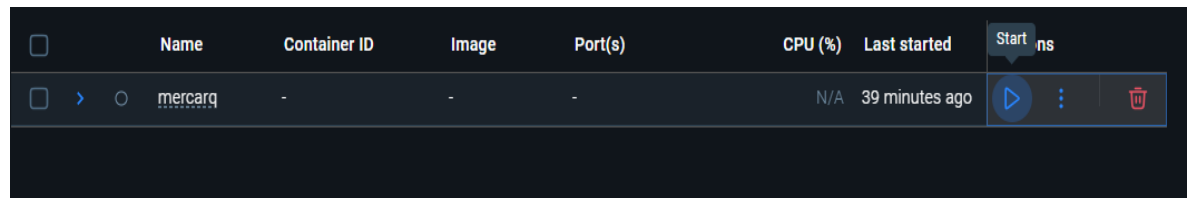
Implementación en .env:

1. se crea un archivo llamado .env
2. Se copia el contenido del archivo .env.example dentro del archivo .env
3. Se configuran las variables necesarias utilizadas en docker:

```
DB_HOST=db
DB_PORT=3306
DB_DATABASE=mercado
DB_USERNAME=root
DB_PASSWORD=secret
```

¿Como levantar el contenedor?:

- Para levantar el docker primero hay que construirlo, y para eso, se puede hacer ambas con el siguiente comando: **docker compose -f docker-compose.yml up - -build**
- La segunda forma solo es posible si trabajas en windows y es que puedes levantar el docker dando click en el contenedor que necesitas. Así como se muestra en la siguiente imagen:



3.2. Modificación Local

Si el desarrollador desea realizar modificaciones del software de manera local, tendrá que realizar la instalación de componentes adicionales, para empezar, se debe de instalar php versión 8 o superiores (para poder trabajar con laragon o Docker), el cual se consigue de manera gratuita en la página <https://www.php.net/downloads.php>, por consiguiente, se deberá de seguir los siguientes pasos

Figura 1. Página web de descarga de PHP

Old Stable PHP 8.2.29 (Changelog)

- [php-8.2.29.tar.gz \(sig\)](#) [18,807Kb] 03 Jul 2025
sha256: 0b27d330769d4bc67b1d8864347c38744b289664a946919c3ddb2235d326b3cd
- [php-8.2.29.tar.bz2 \(sig\)](#) [15,105Kb] 03 Jul 2025
sha256: 51979e8d198cbade2aad4ffe9f53dd3f04f9602d3089e5979985e058ade4267c
- [php-8.2.29.tar.xz \(sig\)](#) [11,877Kb] 03 Jul 2025
sha256: 475f991afd2d5b901fb410be407d929bc00c46285d3f439a02c59e8b6fe3589c
- [Windows downloads](#)

[GPG Keys for PHP 8.2](#)

Old Stable PHP 8.1.33 (Changelog)

- [php-8.1.33.tar.gz \(sig\)](#) [19,470Kb] 03 Jul 2025
sha256: ee33568a0e2be0b722b3f9a88cecc578316b66b25c90cd0a4f3b1a5cdc3cd826
- [php-8.1.33.tar.bz2 \(sig\)](#) [15,188Kb] 03 Jul 2025
sha256: b6553451841c1a569865d7fdc83024621ee4434cd8fbfeb0a31588ac9c70685f
- [php-8.1.33.tar.xz \(sig\)](#) [11,620Kb] 03 Jul 2025
sha256: 9db83bf4590375562bc1a10b353cccbcf9fcfc56c58b7c8fb814e6865bb928d1
- [Windows downloads](#)

[GPG Keys for PHP 8.1](#)

Fuente: (S/f). Php.net.

Al descargar PHP podemos proceder con su debida instalación una vez que hemos instalado y configurado el PHP en las variables de entorno de nuestro equipo, podemos proceder a instalar el editor de texto, el cual para el desarrollo de este proyecto fue realizado con visual Studio Code, nos dirigimos a la pagina de descarga <https://code.visualstudio.com/download> y descargamos la última versión en la pagina oficial, esta descarga es con licencia gratuita

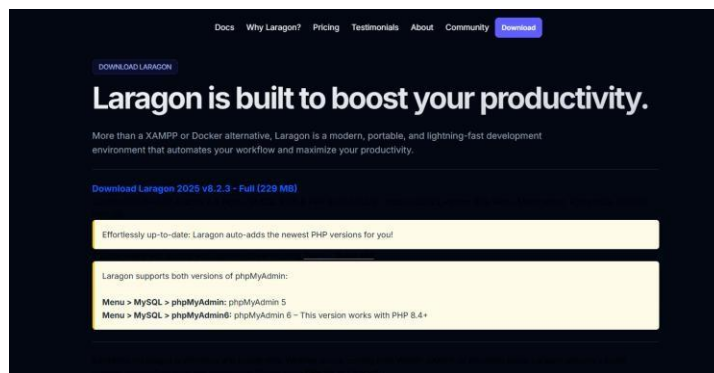
Figura 2 Página de descarga de Visual Studio Code



Fuente: Visual Studio Code. (s/f).

Se continúa descargando el aplicativo de laragon el cual se realiza desde la pagina oficial, laragon es una herramienta de desarrollo específico para Windows el cual brinda todo lo necesario para iniciar el desarrollo en el menor tiempo posible

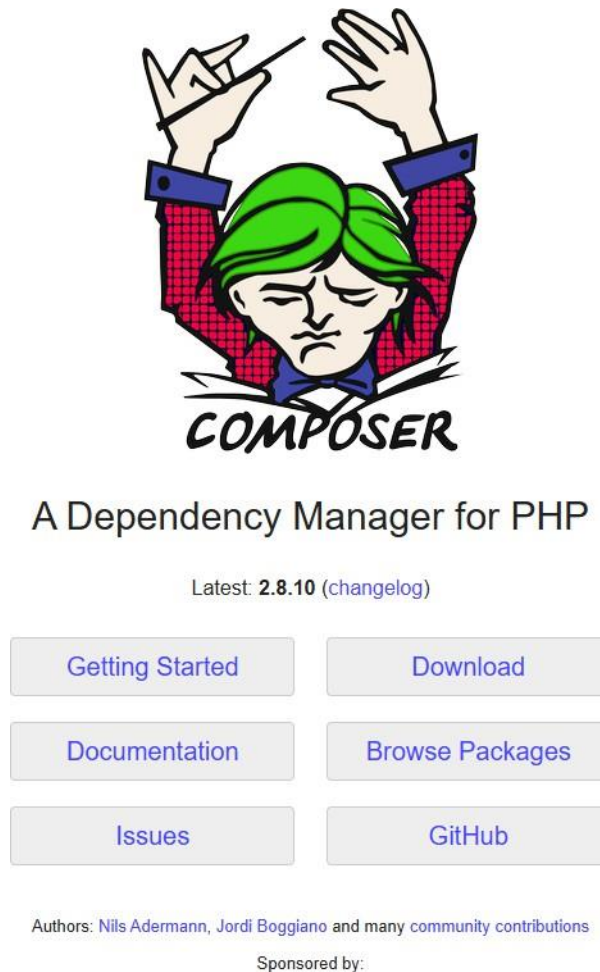
Figura 3 Página de descarga de Laragon



Fuente: LeoKhoa. (s/f). Laragon.

Después de tener PHP, Visual Studio y laragon instalados en nuestro equipo nos hace falta realizar una descarga, la cual es composer, composer es un gestor de paquetes para PHP el cual posibilita la administración, descargas instalaciones y dependencias

Figura 4 Página de descarga de Composer



Fuente: Composer. (s/f).

Teniendo todo lo anterior podemos utilizar git bash para descargar el repositorio del proyecto o hacer un proyecto desde cero, Ejecutando el comando composer create-Project laravel/laravel (Nombre Proyecto)

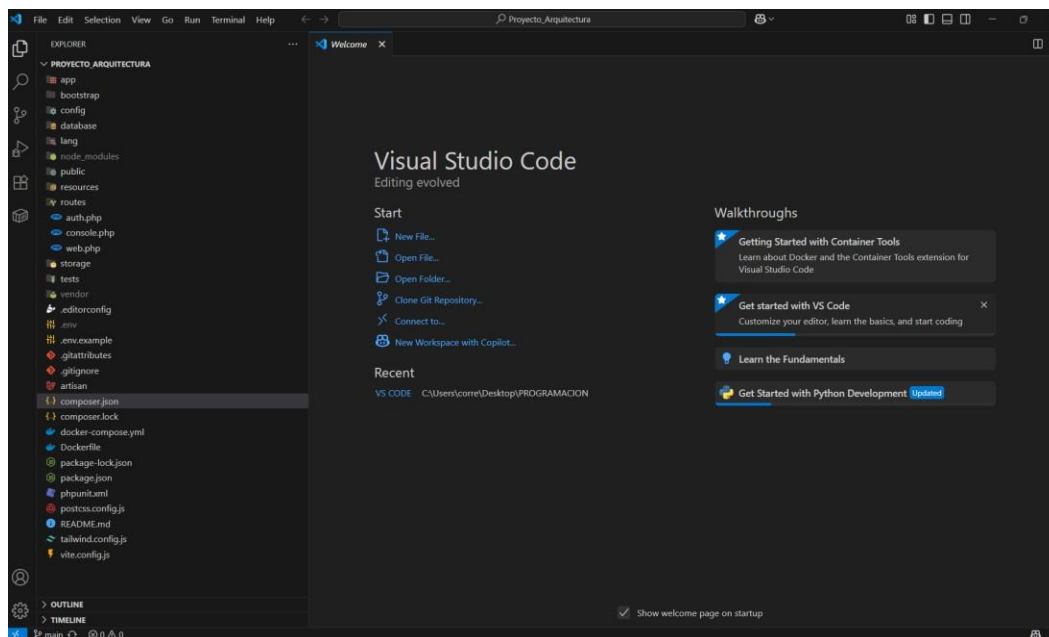
Figura 5 Clonacion repositorio Github

```
$ composer create-project laravel/laravel correcciones_MercarQ
Creating a "laravel/laravel" project at "./correcciones_MercarQ"
Installing laravel/laravel (v12.2.0)
- Installing laravel/laravel (v12.2.0): Extracting archive
Created project in C:\laragon\www\NUEVO PROYECTO\correcciones_MercarQ
> @php -r "file_exists('.env') || copy('.env.example', '.env');"
Loading composer repositories with package information
Updating dependencies
Lock file operations: 110 installs, 0 updates, 0 removals
- Locking brick/math (0.13.1)
- Locking carbonphp/carbon-doctrine-types (3.2.0)
- Locking dflydev/dot-access-data (v3.0.3)
- Locking doctrine/inflector (2.0.10)
- Locking doctrine/lexer (3.0.1)
- Locking dragonmantank/cron-expression (v3.4.0)
- Locking egulias/email-validator (4.0.4)
- Locking fakerphp/faker (v1.24.1)
- Locking filp/whoops (2.18.3)
- Locking fruitcake/php-cors (v1.3.0)
- Locking graham-campbell/result-type (v1.1.3)
- Locking guzzlehttp/guzzle (7.9.3)
- Locking guzzlehttp/promises (2.2.0)
```

Fuente: Por los autores

Una vez realizada la clonación del repositorio de GitHub en nuestra carpeta se debe de proceder con la ejecución del IDE visual studio code en donde a la vez se deberá de seleccionar la carpeta donde hemos clonado el repositorio

Figura 6 Visualización del Proyecto En VSC

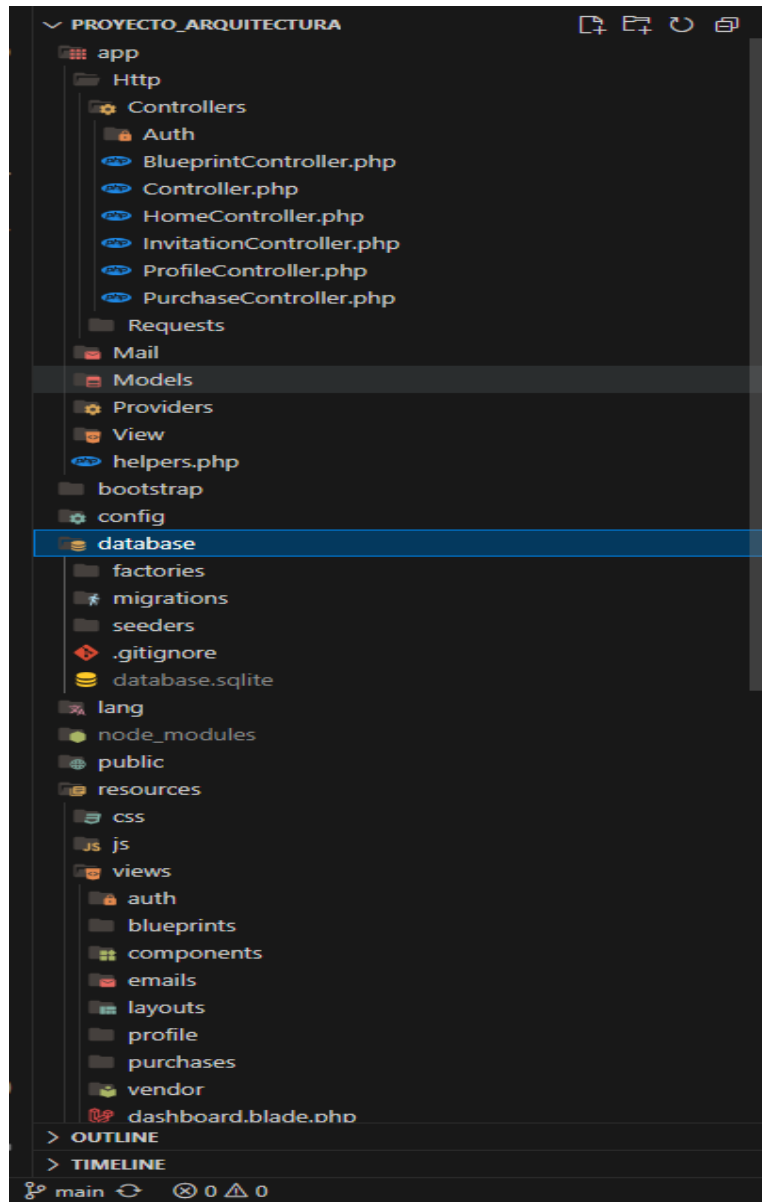


Fuente: Por los autores

Por consiguiente, se nos mostraran una gran variedad de carpetas y programas sin embargo no debe de alterarse, ya que el software solamente utiliza el patrón modelo vista controlador para

este desarrollo eso quiere decir que solo se utilizan las carpetas de controllers model view migrations etc.

Figura 7 Carpetas utilizadas y Patrón MVC



Fuente: Por los autores

El patrón modelo-vista-controlador (MVC) es un patrón de arquitectura de software ampliamente utilizado en el desarrollo de aplicaciones web. Este patrón separa la lógica de la aplicación en tres componentes principales: El modelo, la vista y el controlador. Esta separación de responsabilidades facilita, el desarrollo el mantenimiento y la escalabilidad de las aplicaciones

- **Modelo:** El modelo representa los datos de la aplicación y las reglas de negocio asociadas. Es responsable de la gestión de los datos, como la recuperación, la inserción la actualización y la eliminación. En laravel, los modelos se utilizan para interactuar con la base de datos a través del sistema de eloquent ORM (Object-Relational Mapping)
- **Vista:** La vista es responsable de la presentación de los datos al usuario. Se encarga de generar la interfaz de usuario y mostrar la información recibida desde el Modelo. En laravel, las vistas se crean utilizando archivos de plantillas que contienen una combinación de HTML, CSS y código PHP más conocidas como Blade
- **Controlador:** El controlador actúa como intermediario entre el Modelo y la vista Es responsable de manejar las solicitudes del usuario, recuperar los datos necesarios del modelo y pasarlos a la vista para su presentación, En laravel los controladores son clases PHP que contienen métodos que corresponden a diferentes rutas y acciones de la aplicación.

El uso del patrón modelo vista controlador en Laravel y PHP nos ofrece múltiples beneficios:

- **Separación de responsabilidades:** Mejora la organización y el mantenimiento del código al dividirlo en lógica, interfaz y flujo.
- **Escalabilidad:** Permite integrar nuevas funciones sin afectar el funcionamiento existente.
- **Colaboración en equipo:** Facilita que varios desarrolladores trabajen en paralelo sin conflictos. **Reutilización de código:** Ahorra tiempo y esfuerzo al permitir reaprovechar vistas y controladores
- **Integración con herramientas de Laravel:** Potencia el patrón con funcionalidades como rutas, middleware y migraciones.

Una vez implementadas las mejoras y modificaciones al código se deberá de subir los cambios al repositorio de GitHub, para ello inicializamos nuestra consola de Bash, y agregamos los comandos típicos para subirlos `git init`, `git add .`, `git commit -m "descripción de lo que hiciste"`, `git push origin main` o `master` dependiendo de cual sea tu rama principal si usas main el cambio estará en local y tendrás que desde la plataforma autorizar esta subida haciendo el commit y el merge . Una vez realizados los cambios en la plataforma se nos demostrara quien trabajo que hizo y con qué cambios

Figura 8 Cambios repositorio GitHub

app	proyecto finalizado falta manuales	18 hours ago
bootstrap	Proyecto inicial Mercarq	4 days ago
config	Commit inicial: Proyecto Laravel Merqark	2 days ago
database	proyecto finalizado falta manuales	18 hours ago
lang	Commit inicial: Proyecto Laravel Merqark	2 days ago
public	proyecto finalizado falta manuales	18 hours ago
resources	proyecto finalizado falta manuales	18 hours ago
routes	proyecto finalizado falta manuales	18 hours ago
storage	Proyecto inicial Mercarq	4 days ago
tests	Implementa funcionalidad de planos, compras, perfil y dash...	4 days ago
.editorconfig	Proyecto inicial Mercarq	4 days ago
.env.example	Se modifica para poder trabajar en tiempo real	yesterday
.gitattributes	Proyecto inicial Mercarq	4 days ago
.gitignore	Proyecto inicial Mercarq	4 days ago
Dockerfile	Se modifica para poder trabajar en tiempo real	yesterday
README.md	Proyecto inicial Mercarq	4 days ago
artisan	Proyecto inicial Mercarq	4 days ago

Fuente: Por los autores

Figura 9 Validación Composer.Json

```

1  {
2    "$schema": "https://getcomposer.org/schema.json",
3    "name": "laravel/laravel",
4    "type": "project",
5    "description": "The skeleton application for the Laravel framework.",
6    "keywords": ["laravel", "framework"],
7    "license": "MIT",
8    "require": {
9      "php": "^8.2",
10     "laravel/framework": "^12.0",
11     "laravel/tinker": "^2.10.1"
12   },
13   "require-dev": {
14     "fakerphp/faker": "^1.23",
15     "laravel/breeze": "^2.3",
16     "laravel/pail": "^1.2.2",
17     "laravel/pint": "^1.13",
18     "laravel/sail": "^1.41",
19     "mockery/mockery": "^1.6",
20     "nunomaduro/collision": "^8.6",
21     "pestphp/pest": "^3.8",
22     "pestphp/pest-plugin-laravel": "^3.2"
23   },
24   "autoload": {
25     "psr-4": {
26       "App\\": "app/",
27       "Database\\Factories\\": "database/factories/",
28       "Database\\Seeders\\": "database/seeders/"
29     }
30   },
31   "autoload-dev": {
32     "psr-4": {
33       "Tests\\": "tests/"
34     }
35   },
36   "scripts": {
37     "post-autoload-dump": [
38       "Illuminate\\Foundation\\ComposerScripts::postAutoloadDump",
39       "@php artisan package:discover --ansi"
40     ],
41     "post-update-cmd": [
42       "@php artisan vendor:publish --tag=laravel-assets --ansi --force"
43     ],
44     "post-root-package-install": [
45       "@php -r \"file_exists('.env') || copy('.env.example', '.env');\""
46     ],
47     "post-create-project-cmd": [
48       "@php artisan key:generate --ansi",

```

Fuente: Por los autores

El archivo composer.json en Laravel es como el corazón de nuestro proyecto. Es un archivo que contiene la información sobre todas y cada una de las dependencias del proyecto, es decir, los otros paquetes de código que el proyecto necesita para funcionar correctamente. También contiene información de configuración y se utiliza para gestionar la autocarga de clases

Figura 10 inicialización en servidor local

```

PS C:\laragon\www\Proyecto_Arquitectura> php artisan serve

INFO  Server running on [http://127.0.0.1:8000].

Press Ctrl+C to stop the server

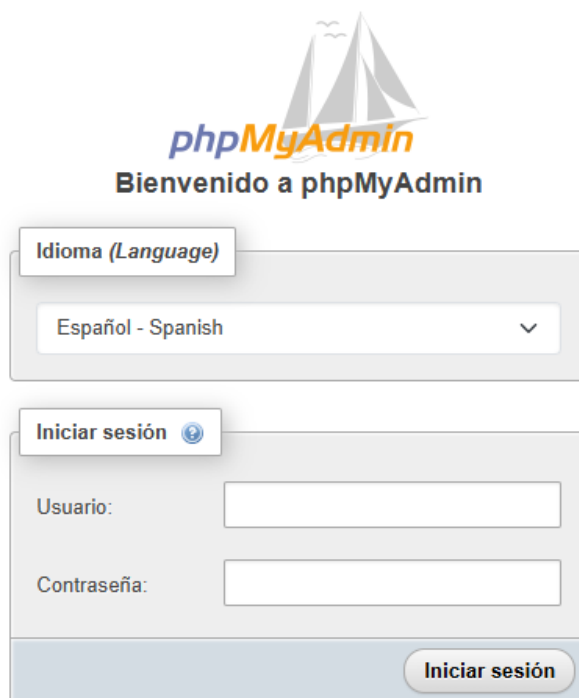
2025-07-30 14:17:22 / ..... ~ 9s
2025-07-30 14:17:32 /favicon.ico ..... ~ 0.92ms

```

Fuente: Por los autores

Existen 2 maneras la 1 es utilizando la manera local de nuestro vscode el comando php artisan serve y ejecutar el link que nos aparece, la 2 es abrir laragon y navegar hacia la carpeta donde tenemos alojado el proyecto

Figura 11 Ingreso a la administración de PhpMyAdmin



The image shows the PhpMyAdmin login interface. At the top, there is a logo with a sailboat and the text "phpMyAdmin" and "Bienvenido a phpMyAdmin". Below this, there is a language selection dropdown menu labeled "Idioma (Language)" with "Español - Spanish" selected. Underneath is a login form with fields for "Usuario:" and "Contraseña:", and a button labeled "Iniciar sesión".

Fuente: Por los autores

Al ingresar al gestor de bases de datos PhpMyAdmin se podrá tener la gerencia completa de nuestras tablas en caso de necesitar la inserción creación o modificación de una tabla

Figura 12 Administración en PhpMyAdmin

Que contengan la palabra:

Tabla	Acción	Filas	Tipo	Cotejamiento	Tamaño	Residuo a depurar
<input type="checkbox"/> blueprints	★ Examinar Estructura Buscar Insertar Vaciar Eliminar	8	InnoDB	utf8mb4_unicode_ci	32.0 KB	-
<input type="checkbox"/> cache	★ Examinar Estructura Buscar Insertar Vaciar Eliminar	0	InnoDB	utf8mb4_unicode_ci	16.0 KB	-
<input type="checkbox"/> cache_locks	★ Examinar Estructura Buscar Insertar Vaciar Eliminar	0	InnoDB	utf8mb4_unicode_ci	16.0 KB	-
<input type="checkbox"/> failed_jobs	★ Examinar Estructura Buscar Insertar Vaciar Eliminar	0	InnoDB	utf8mb4_unicode_ci	16.0 KB	-
<input type="checkbox"/> jobs	★ Examinar Estructura Buscar Insertar Vaciar Eliminar	0	InnoDB	utf8mb4_unicode_ci	16.0 KB	-
<input type="checkbox"/> job_batches	★ Examinar Estructura Buscar Insertar Vaciar Eliminar	0	InnoDB	utf8mb4_unicode_ci	16.0 KB	-
<input type="checkbox"/> migrations	★ Examinar Estructura Buscar Insertar Vaciar Eliminar	8	InnoDB	utf8mb4_unicode_ci	16.0 KB	-
<input type="checkbox"/> password_reset_tokens	★ Examinar Estructura Buscar Insertar Vaciar Eliminar	0	InnoDB	utf8mb4_unicode_ci	16.0 KB	-
<input type="checkbox"/> purchases	★ Examinar Estructura Buscar Insertar Vaciar Eliminar	0	InnoDB	utf8mb4_unicode_ci	48.0 KB	-
<input type="checkbox"/> sessions	★ Examinar Estructura Buscar Insertar Vaciar Eliminar	5	InnoDB	utf8mb4_unicode_ci	48.0 KB	-
<input type="checkbox"/> solicitudes	★ Examinar Estructura Buscar Insertar Vaciar Eliminar	6	InnoDB	utf8mb4_unicode_ci	32.0 KB	-
<input type="checkbox"/> users	★ Examinar Estructura Buscar Insertar Vaciar Eliminar	3	InnoDB	utf8mb4_unicode_ci	32.0 KB	-
12 tablas	Número de filas	30	InnoDB	utf8mb4_unicode_ci	304.0 KB	0 B

☐ Seleccionar todo
 Para los elementos que están marcados:

Fuente: Por los autores

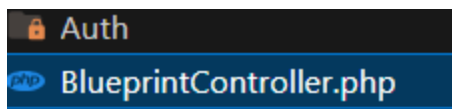
Explicación Código

Controladores:

En la carpeta de app/http/controllers se encuentra cada dirección específica de los controladores

BlueprintController:

Figura 13 BlueprintController

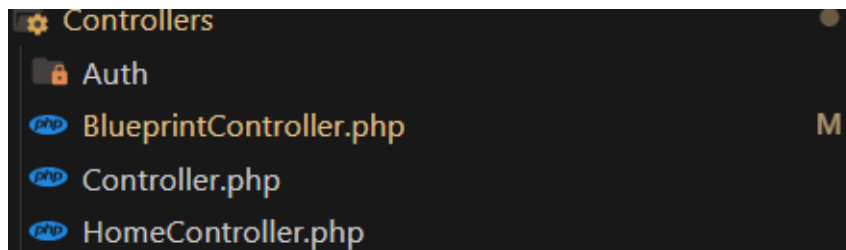


Fuente: Por los autores

El controlador BlueprintController gestiona integralmente el ciclo de vida de los planos técnicos (blueprints) en la aplicación, proporcionando funcionalidad completa para la gestión de archivos técnicos con soporte para carga, edición, visualización pública/privada y procesos de descarga/compra. Este controlador implementa patrones RESTful estándar en Laravel donde los métodos index(), create(), store(), show(), edit(), update() y destroy() manejan las operaciones CRUD básicas con validaciones estrictas de archivos (permitiendo formatos como PDF, DWG, DXF e imágenes) y seguridad mediante verificación de propiedad del recurso antes de cualquier acción modificadora. Las funcionalidades especializadas incluyen el sistema de notificación por WhatsApp (whatsappPurchase()) que notifica al vendedor cuando un comprador muestra interés, y los flujos de descarga gratuita (freeDownload() y downloadFree()) que gestionan tanto solicitudes con formulario como descargas directas mientras registran actividades en el modelo Solicitud y notifican por correo electrónico a ambas partes mediante clases Mailable personalizadas. El controlador también implementa capas de autorización rigurosas mediante Auth::id() para proteger recursos privados, gestiona almacenamiento eficiente en el disco público con eliminación segura de archivos antiguos durante actualizaciones, y ofrece búsquedas avanzada en publicIndex() con logging detallado para auditoría, todo siguiendo las mejores prácticas de Laravel para validación de formularios, manejo de archivos y gestión de relaciones modelo-controlador.

HomeController:

Figura 14 HomeController

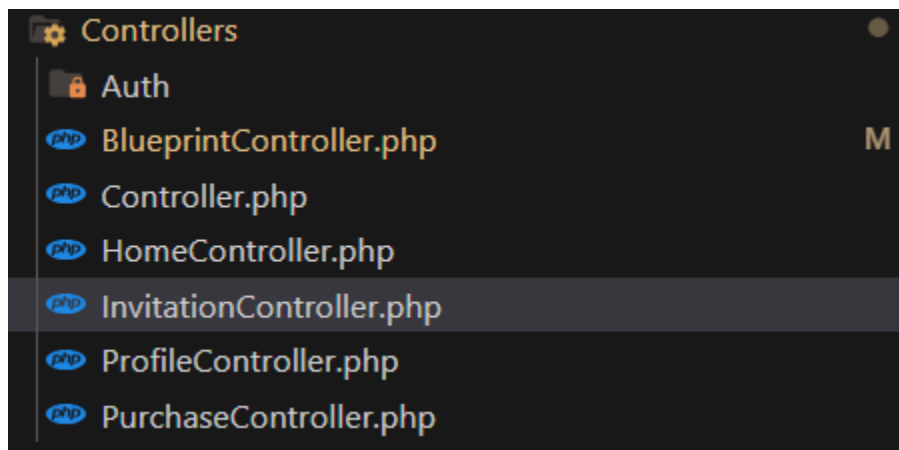


Fuente: Por los autores

El controlador HomeController gestiona la lógica del sitio principal de la aplicación, específicamente su método index() recupera los 6 planos técnicos más recientes marcados como públicos (utilizando Blueprint::where('is_public', true)->latest()->limit(6)) para mostrarlos en la página de inicio, garantizando que solo se exponga contenido autorizado mediante el filtro is_public y priorizando la frescura de los datos con latest(). Este enfoque optimiza la experiencia del usuario al cargar de forma eficiente una selección destacada de recursos técnicos directamente en la vista index.blade.php mediante compact('latestBlueprints'), cumpliendo con el patrón de diseño MVC al separar claramente la lógica de negocio (consulta a la base de datos) de la presentación (renderizado de la vista), y sirviendo como punto de entrada para visitantes que exploran planos disponibles públicamente sin requerir autenticación.

InvitationController

Figura 15 InvitationController

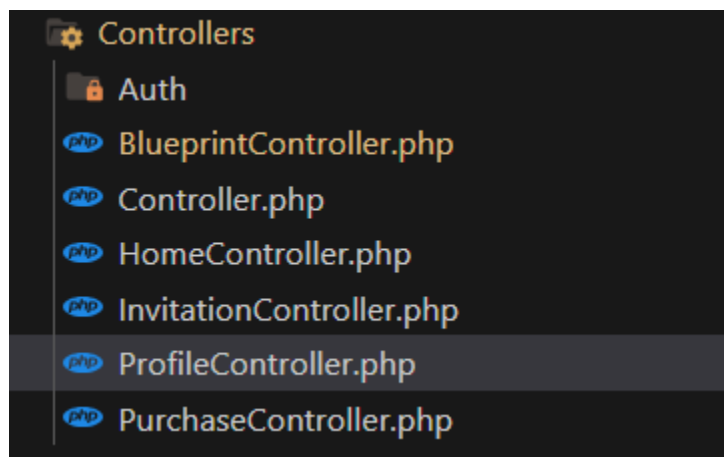


Fuente: Por los autores

El controlador InvitationController gestiona el proceso de invitación y registro directo de nuevos usuarios mediante un flujo seguro y validado, donde el método showInvitationForm() carga la interfaz de invitación (auth.invite.blade.php), mientras que inviteUser() procesa los datos con validaciones rigurosas: asegura que el correo sea único (evitando duplicados), aplica políticas de contraseña robustas mediante Rules\Password::defaults(), y encripta la contraseña con Hash::make() siguiendo estándares de seguridad de Laravel.

ProfileController

Figura 16 ProfileController

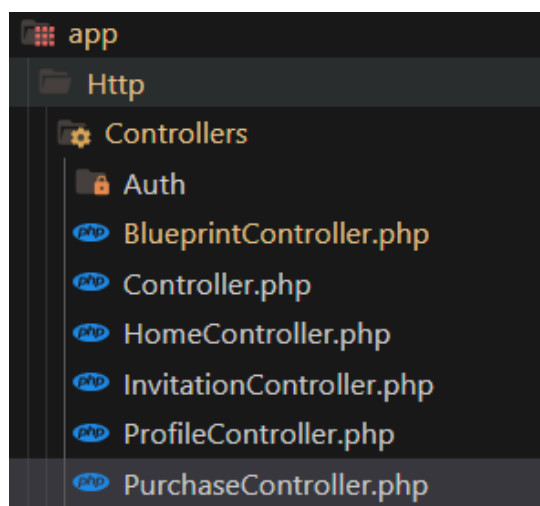


Fuente: Por los autores

El controlador ProfileController gestiona de manera segura y eficiente la edición, actualización y eliminación de perfiles de usuario en la aplicación, implementando flujos alineados con las mejores prácticas de Laravel para autenticación y gestión de datos personales. En el método update(), procesa solicitudes validadas mediante ProfileUpdateRequest (garantizando integridad de datos), maneja la carga de avatares con lógica robusta: elimina la imagen anterior si existe, almacena el nuevo archivo en el disco público bajo el directorio avatars/, y actualiza la ruta en la base de datos, todo mientras respeta el principio de single responsibility al delegar validaciones a clases específicas

PurchaseController

Figura 17 PurchaseController



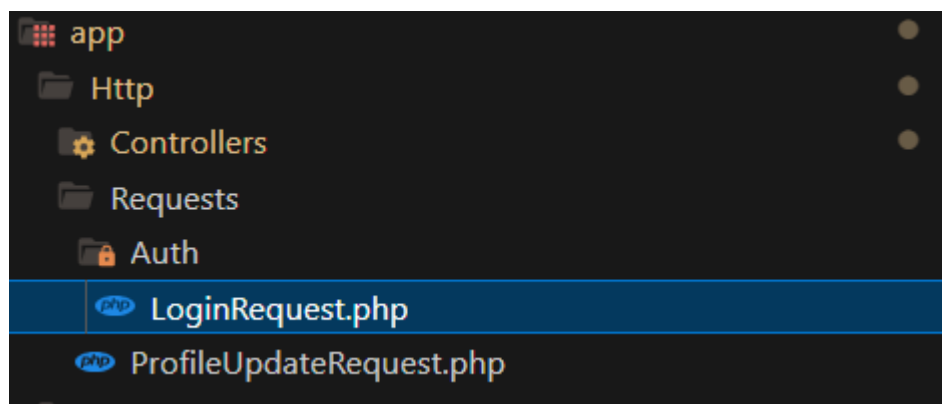
Fuente: Por los autores

El controlador PurchaseController gestiona dos flujos críticos en el sistema de transacciones: la visualización de solicitudes de contacto por parte de vendedores y la consulta detallada de compras por compradores, implementando autorización estricta y relaciones modelo-datos optimizadas. En su método index(), recupera todas las Solicitud (interacciones de potenciales compradores) asociadas a los planos técnicos del usuario autenticado mediante whereHas('blueprint'), asegurando que solo los vendedores vean solicitudes relevantes a sus propios planos mediante la condición where('user_id', \$user->id), mientras aplica paginación y carga ansiosa (with('blueprint')) para optimizar el rendimiento y evitar problemas de N+1. Por otro lado, el método show() está diseñado exclusivamente para compradores, verificando que la Purchase (transacción completada) pertenezca al usuario actual mediante Auth::id(), y cargando datos relacionados como blueprint. User para mostrar información del vendedor en la vista, garantizando encapsulación de datos sensibles mediante el aborto 403 en accesos no autorizados.

Solicitudes – Request

LoginRequest

Figura 18 LoginRequest

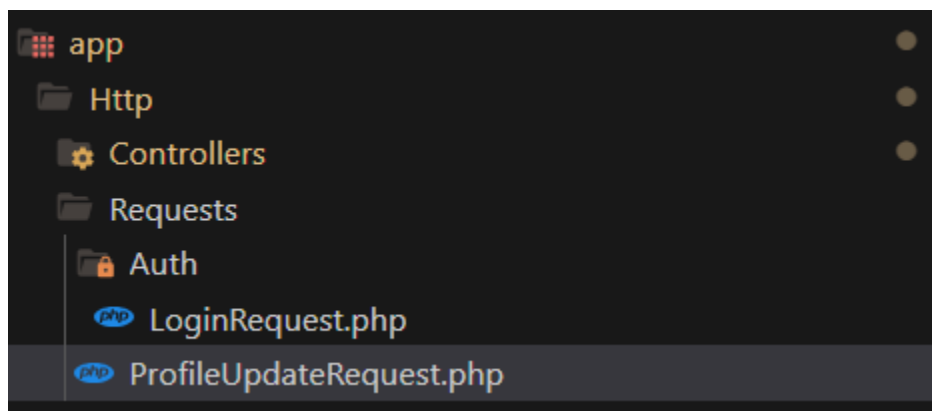


Fuente: Por los autores

La clase LoginRequest es un Form Request personalizado de Laravel que maneja de forma integral el proceso de autenticación de usuarios en el sistema. Su función principal es validar las credenciales de login (email y contraseña) y ejecutar el proceso de autenticación de manera segura, incorporando un sistema de protección contra ataques de fuerza bruta mediante rate limiting. La clase valida que el email tenga formato correcto y que la contraseña esté presente, luego intenta autenticar al usuario usando las credenciales proporcionadas. Si las credenciales son incorrectas, incrementa un contador de intentos fallidos asociado a la combinación de email e IP del usuario; si se superan 5 intentos fallidos, bloquea temporalmente nuevos intentos de login desde esa combinación específica, disparando un evento Lockout y mostrando un mensaje con el tiempo restante para poder intentar nuevamente.

ProfileUpdateRequest

Figura 19 ProfileUpdateRequest

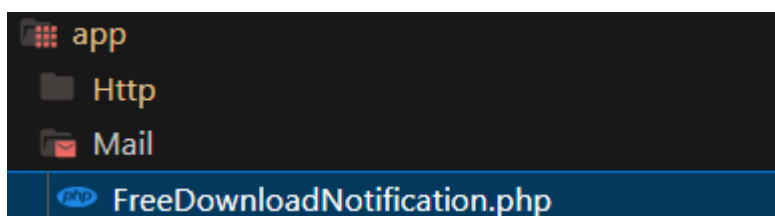


Fuente: Por los autores

La clase ProfileUpdateRequest define las reglas de validación específicas para actualizar datos del perfil de usuario en la aplicación, garantizando que los datos ingresados cumplan con estándares de seguridad e integridad antes de procesarlos en el controlador. En su método rules(), implementa validaciones estrictas: el campo name es obligatorio, debe ser una cadena de máximo 255 caracteres, mientras que el email requiere formato válido, conversión automática a minúsculas (lowercase), y unicidad en la base de datos exceptuando el ID del usuario actual mediante Rule::unique(User::class)->ignore(\$this->user()->id), lo que evita errores de duplicado al editar el propio perfil sin cambiar el correo

FreeDownloadOwnerNotification

Figura 20 FreeDownloadOwnerNotification

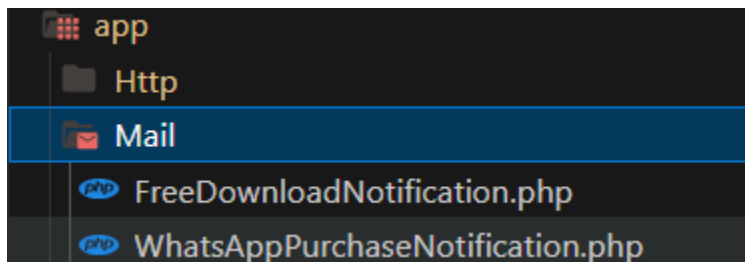


Fuente: Por los autores

Es un Mailable de Laravel que se encarga de notificar automáticamente al propietario de un plano (blueprint) cuando alguien descarga gratuitamente su diseño desde la plataforma. La clase recibe como parámetros el objeto Blueprint descargado y los datos del usuario que realizó la descarga (nombre y email), y en su constructor registra automáticamente esta actividad en la base de datos creando un nuevo registro en la tabla solicitudes con tipo "descarga_gratuita", almacenando información como el ID del plano, nombre y email del descargador, y la dirección IP desde donde se realizó la descarga. El sistema incluye logging detallado para monitorear el proceso y capturar cualquier error que pueda ocurrir durante el registro

WhatsAppPurchaseNotification

Figura 21 WhatsAppPurchaseNotification



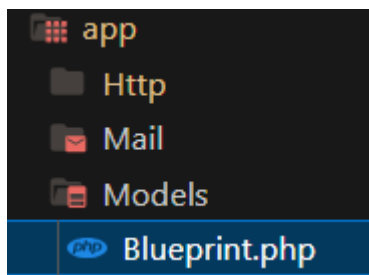
Fuente: Por los autores

La clase WhatsAppPurchaseNotification implementa un sistema de notificación automatizada para vendedores de planos técnicos que se activa cuando un comprador muestra interés en adquirir un plano mediante WhatsApp, cumpliendo una doble función crítica: primero, registra automáticamente la interacción en la base de datos mediante el modelo Solicitud (almacenando ID del plano, datos del comprador, mensaje e IP para auditoría), y segundo, genera un correo estructurado con un enlace preconfigurado de WhatsApp que incluye un mensaje inicial personalizado con detalles del plano y precio formateado, facilitando la comunicación inmediata entre partes

Modelos – Models

Blueprint

Figura 22 Modelo Blueprint

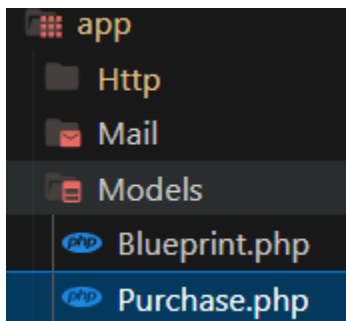


Fuente: Por los autores

El modelo Blueprint representa la estructura central de los planos técnicos en la aplicación, definiendo sus propiedades esenciales mediante atributos \$fillable como title, file_path, price e is_public, y aplicando transformaciones automáticas críticas a través de \$casts: convierte price a decimal con dos decimales para precisión financiera, is_public a booleano para lógica de acceso, y file_size a entero para manejo consistente de tamaño de archivos.

Purchase

Figura 23 Modelo Purchase

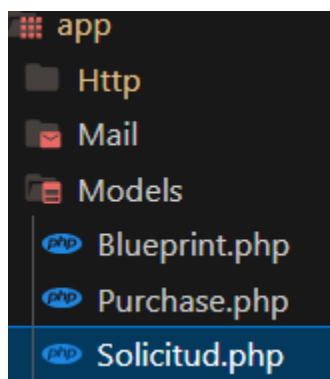


Fuente: Por los autores

Es un modelo Eloquent de Laravel que representa las transacciones de compra de planos (blueprints) realizadas por los usuarios en la plataforma. El modelo gestiona toda la información relacionada con las compras, incluyendo qué usuario realizó la compra (`user_id`), qué plano fue adquirido (`blueprint_id`), el monto pagado (`amount`), el método de pago utilizado (`payment_method`), la referencia o ID de la transacción del procesador de pagos (`payment_reference`), y el estado actual de la compra (`status`) que puede indicar si está pendiente, completada, fallida, etc. La clase establece las relaciones necesarias con otros modelos a través de dos métodos: `user()` que conecta cada compra con el usuario que la realizó, y `blueprint()` que la vincula con el plano específico que fue adquirido.

Solicitud

Figura 24 Modelo Solicitud



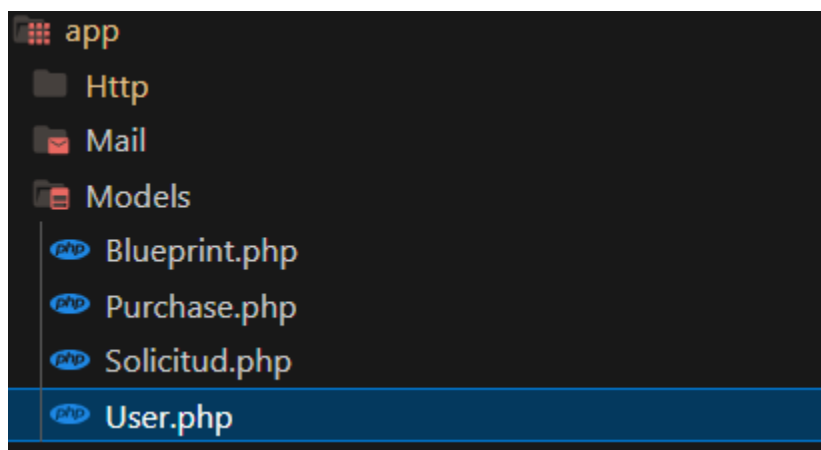
Fuente: Por los autores

El modelo Solicitud representa un registro histórico de todas las interacciones de usuarios con los planos técnicos, almacenando metadatos críticos para el seguimiento comercial y auditoría, donde cada instancia se asocia a una tabla personalizada solicitudes (en lugar de seguir la convención

plural de Laravel) mediante `protected $table = 'solicitudes'`. Define campos esenciales en `$fillable` que capturan el contexto completo de cada interacción: `blueprint_id` (relación con el plano), `tipo_solicitud` (ej.: 'whatsapp' o 'descarga_gratuita'), datos del interesado (`nombre_solicitante`, `email_solicitante`, `telefono_solicitante`), mensaje personalizado y `ip_address` para seguridad, permitiendo distinguir claramente entre flujos de compra y descarga gratuita mediante el campo `tipo_solicitud`.

User

Figura 25 Modelo User



Fuente: Por los autores

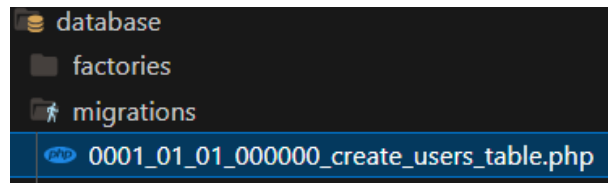
Esta clase es el modelo Eloquent central que representa a los usuarios de la plataforma, extendiendo la clase `Authenticatable` de Laravel para proporcionar funcionalidades completas de autenticación y autorización. El modelo gestiona información básica del usuario como nombre, email, contraseña, rol del sistema y ruta del avatar, implementando medidas de seguridad como el ocultamiento automático de la contraseña y token de recordatorio, además del hash automático de contraseñas. Incluye un sistema de roles con el método `isAdmin()` para verificar privilegios administrativos, y maneja avatares de usuario mediante el accessor `getAvatarUrlAttribute()` que devuelve la imagen personalizada del usuario o genera automáticamente un avatar con iniciales usando el servicio UI Avatars como fallback.

Migraciones- Migrations

Se encuentran en la carpeta `database/migrations`

Migración Crear Usuarios

Figura 26 Migración crear Usuarios

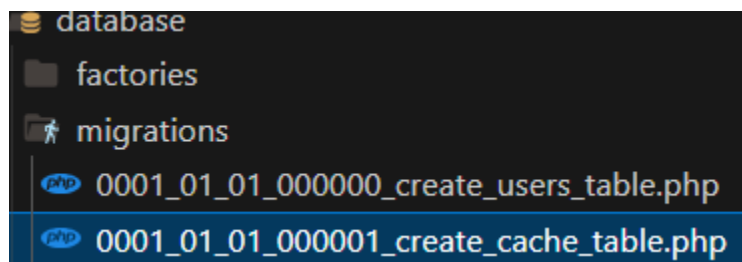


Fuente: Por los autores

define la estructura inicial de la base de datos para gestionar usuarios y autenticación en la aplicación, implementando tres tablas esenciales con enfoque en seguridad y escalabilidad. La tabla users incluye campos críticos como email (único para evitar duplicados), password (almacenado encriptado), y un campo role con valor predeterminado 'cliente' que permite futuras implementaciones de roles de autorización (como vendedores o administradores), mientras que los timestamps automáticos (created_at, updated_at) facilitan el seguimiento de actividad.

Migración create_cache

Figura 27 Migración create_cache

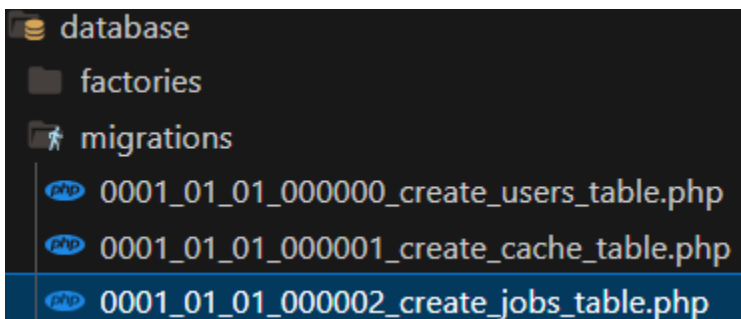


Fuente: Por los autores

Esta migración de Laravel establece la infraestructura necesaria para el sistema de caché basado en base de datos, creando dos tablas fundamentales para el almacenamiento y gestión del caché. La tabla cache actúa como el repositorio principal donde se almacenan los datos en caché, utilizando una clave primaria de tipo string (key) como identificador único, un campo value de tipo mediumText para almacenar el contenido serializado del caché (que puede incluir objetos, arrays, strings, etc.), y un campo expiration de tipo integer que registra el timestamp Unix cuando el elemento del caché debe expirar.

Migración Create_Jobs

Figura 28 Migración Create_Jobs

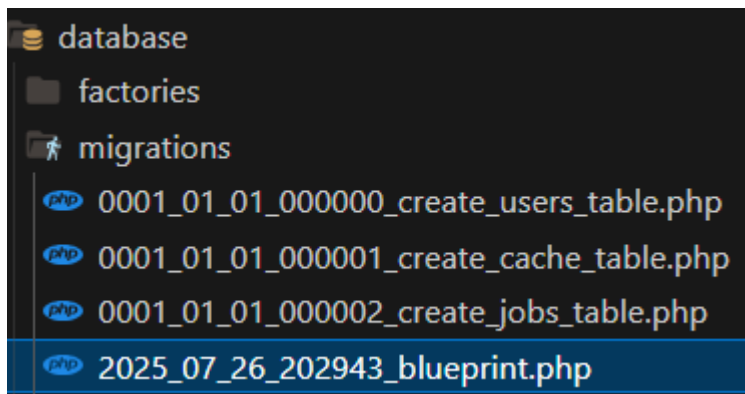


Fuente: Por los autores

define la estructura de base de datos para el sistema de colas de Laravel, esencial para procesar tareas asíncronas y mejorar la escalabilidad de la aplicación, implementando tres tablas interconectadas: jobs almacena trabajos pendientes con metadatos como payload (datos serializados), attempts (intentos de procesamiento) y tiempos de disponibilidad (available_at), permitiendo ejecutar operaciones intensivas (como envío de correos de notificación WhatsApp) sin bloquear la interfaz; job_batches gestiona lotes de trabajos relacionados mediante campos como total_jobs, pending_jobs y failed_jobs, facilitando el seguimiento de procesos complejos como la generación masiva de enlaces;

Migración create_blueprints

Figura 29 Migración create_blueprints



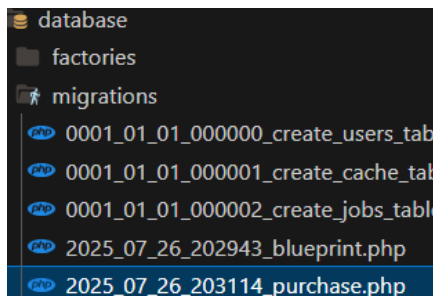
Fuente: Por los autores

Crea la tabla blueprints, que constituye el núcleo del sistema de gestión de planos y diseños de la plataforma. La tabla almacena información completa sobre cada plano subido por los usuarios, incluyendo un identificador único autoincremental (id), una referencia al usuario propietario (user_id) con restricción de clave foránea y eliminación en cascada para mantener la integridad referencial, título y descripción opcional del plano (title, description), y la ruta del archivo físico en el sistema de almacenamiento (file_path). El sistema incorpora funcionalidades comerciales mediante el campo price que define el costo del plano en formato entero (centavos), un número

de WhatsApp (whatsapp_number) para contacto directo con el creador, y el tamaño del archivo (file_size) para optimización de descargas

Migración Purchase

Figura 30 Migración Purchase

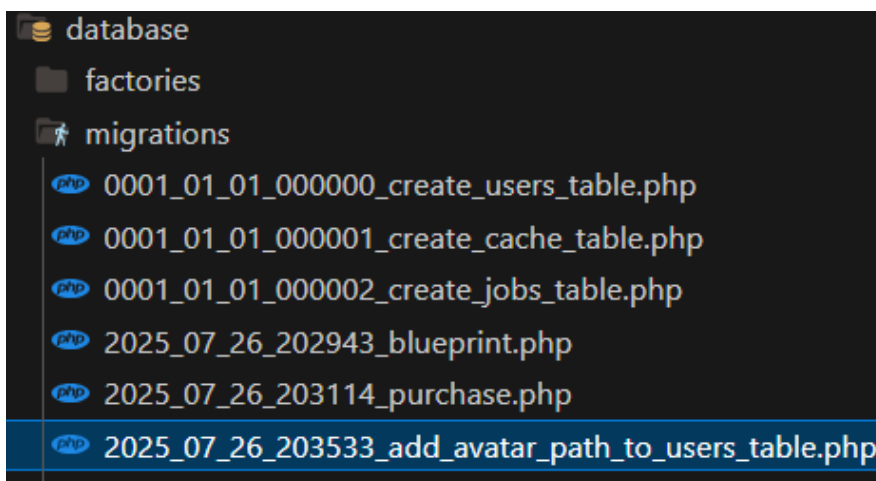


Fuente: Por los autores

Define la estructura de la tabla transaccional central que registra todas las compras de planos técnicos en la aplicación, estableciendo relaciones críticas mediante claves foráneas con eliminación en cascada (onDelete('cascade')): user_id vincula al comprador (tabla users) y blueprint_id al plano adquirido (tabla blueprints), garantizando que al eliminar un usuario o plano, sus transacciones asociadas se limpien automáticamente para mantener integridad referencial. El campo amount almacena el monto con precisión decimal (10,2), esencial para cálculos financieros exactos, mientras payment_method y payment_reference capturan detalles del proceso de pago (aunque actualmente el sistema parece priorizar notificaciones por WhatsApp sobre pasarelas de pago tradicionales, como evidencia el valor predeterminado status = 'completed' y la ausencia de estados intermedios como 'pending').

Migración add_avatar_path

Figura 31 add_avatar_path

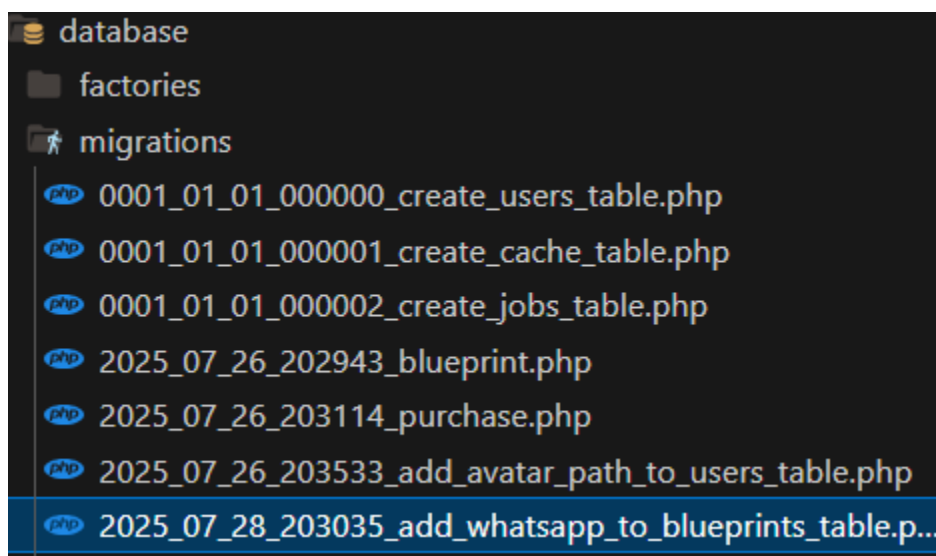


Fuente: Por los autores

modifica la tabla existente users para incorporar funcionalidades de personalización de perfil mediante la adición del campo avatar_path. El nuevo campo de tipo string nullable se posiciona estratégicamente después del campo email y está diseñado para almacenar la ruta relativa o nombre del archivo de imagen que servirá como avatar o foto de perfil del usuario. Al ser nullable, el sistema permite que los usuarios operen sin necesidad de subir obligatoriamente una imagen de perfil, proporcionando flexibilidad en la experiencia de usuario. Esta funcionalidad es esencial para humanizar la interfaz de la plataforma, especialmente en un sistema donde los usuarios interactúan mediante la compra y venta de planos, ya que facilita la identificación visual de los creadores

Migración add_whatsapp_Blueprints

Figura 32 add_whatsapp_Blueprints

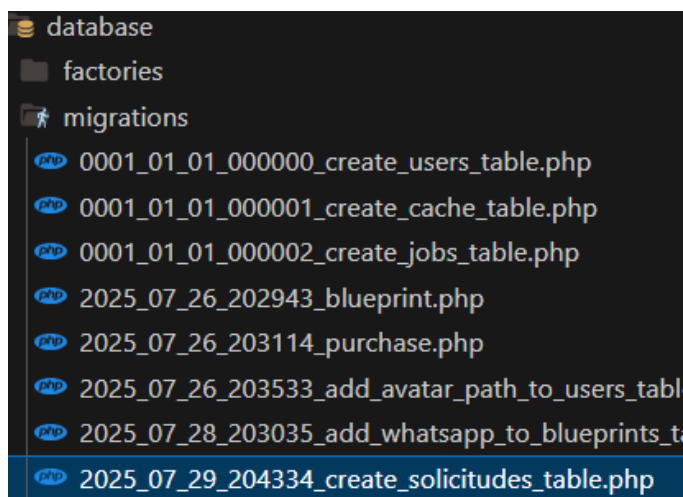


Fuente: Por los autores

corresponde a un archivo de modificación vacío para la tabla blueprints, diseñado específicamente como plantilla para futuras actualizaciones del esquema sin realizar cambios inmediatos en la base de datos. Su estructura básica utiliza Schema::table('blueprints', ...) en lugar de Schema::create(), indicando claramente que está destinada a alterar una tabla existente (previamente creada en otra migración) en caso de que ocurran errores

Migración create_solicitudes

Figura 33 create_solicitudes

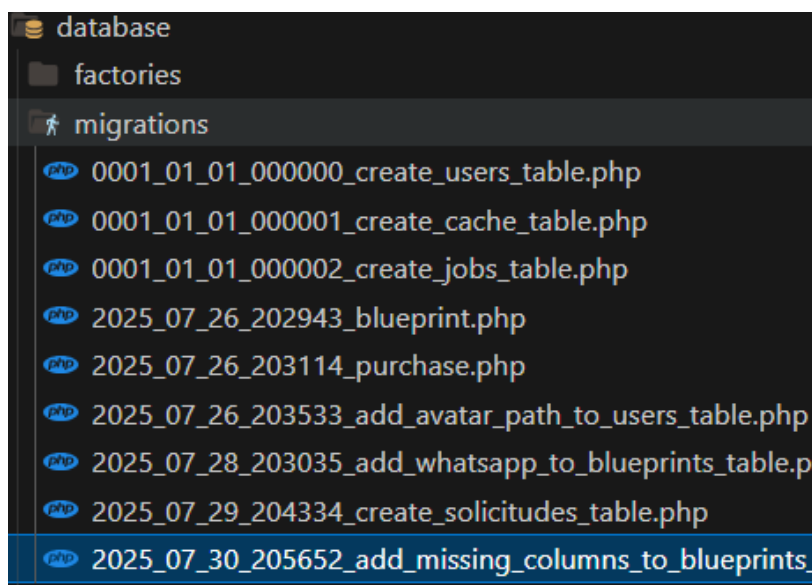


Fuente: Por los autores

crea la tabla solicitudes, que funciona como un sistema centralizado de registro y seguimiento de todas las interacciones y solicitudes relacionadas con los planos de la plataforma. La tabla está diseñada para capturar diversos tipos de actividades mediante el campo tipo_solicitud que puede almacenar valores como "descarga_gratuita", "consulta", "cotización", etc., permitiendo categorizar y filtrar diferentes tipos de interacciones. Incluye una referencia opcional al plano involucrado (blueprint_id) con restricción de clave foránea configurada para establecer null en caso de eliminación del plano, preservando el historial de actividades incluso si el contenido original se elimina. La tabla almacena información completa del solicitante incluyendo nombre obligatorio (nombre_solicitante), email y teléfono opcionales para facilitar el seguimiento

Migración add_missing_columns

Figura 34 add_missing_columns



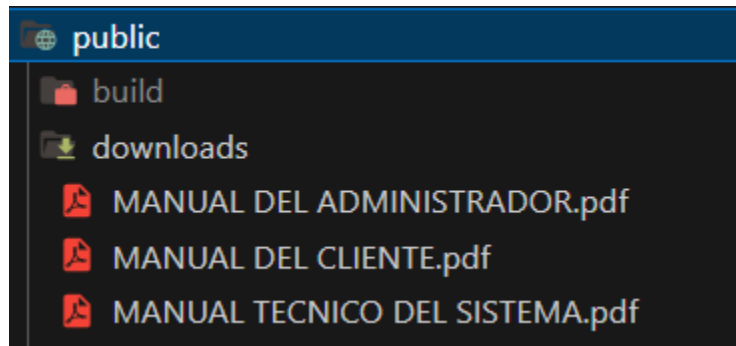
Fuente: Por los autores

La migración modifica la tabla blueprints para soportar la funcionalidad de contacto por WhatsApp y gestión de tamaño de archivos, añadiendo dos columnas esenciales con verificación de existencia previa para evitar conflictos: whatsapp_number (cadena de 20 caracteres) almacena el número de contacto del vendedor utilizado en WhatsAppPurchaseNotification para generar enlaces wa.me, mientras file_size (BigInteger nullable) registra el tamaño en bytes del archivo técnico para su presentación formateada mediante el accessor getFormattedFileSizeAttribute del modelo.

MANUALES

Cada manual se encuentra dentro de la carpeta public/downloads/(nombre manual), el proyecto cuenta con 3 manuales diferentes usuarios no registrados aquellos que no pueden usar el 100% de la plataforma y que en la ocasión solo pueden tener contacto con el arquitecto y unos planos, manual registrados o administrador son los usuarios que pueden acceder al 100% del contenido de la plataforma web en el se especifican todas las funciones que se pueden realizar, si se realiza una nueva modificación a la página web se deberá de actualizar dichos manuales y este ultimo el manual técnico es para cada usuario o miembro del equipo de desarrollo que intente realizar la escalabilidad del proyecto

Figura 35 Manuales del Sistema

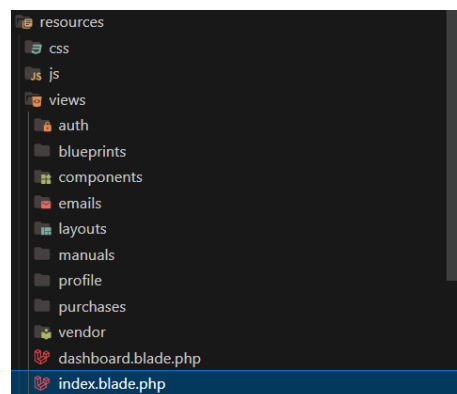


Fuente: Por los autores

Vistas-Views

Son una de nuestras partes esenciales de nuestro programa, es lo que observa el usuario, cada vista esta separada por carpetas, para identificar cada una las vistas del login se encuentran en la parte de auth, los planos en blueprints, barra de navegación entre otros componentes se encuentran en layouts para acceder a estos componentes debes de ubicar la carpeta resources/views

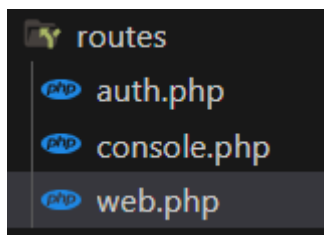
Figura 36 Vistas



Fuente: Por los autores

RUTAS – Routes

Figura 37 Routes



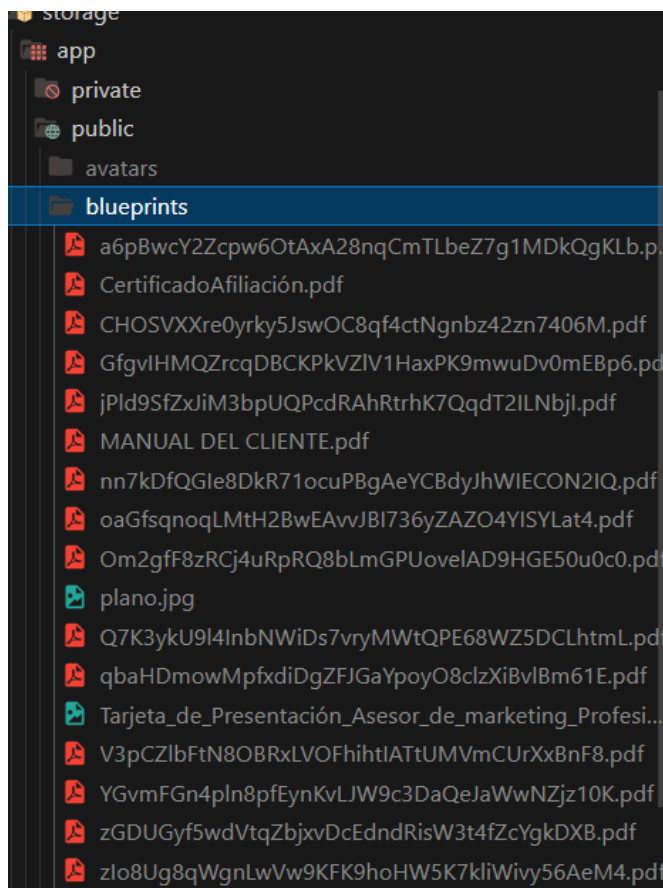
Fuente: Por los autores

Los archivos de rutas en especial los de web.php definen la arquitectura completa de navegación de la plataforma, organizando las URLs en dos secciones principales: rutas públicas accesibles sin autenticación y rutas protegidas que requieren usuario autenticado y verificado. La sección pública incluye la página de inicio (/), el catálogo de planos públicos (/planos) con visualización individual, sistema de descarga gratuita, sección de manuales, y funcionalidades de compra via WhatsApp que permiten a visitantes anónimos interactuar con el contenido básico. Las rutas públicas implementan validaciones de seguridad como verificar que un plano sea público antes de mostrarlo, evitando acceso no autorizado a contenido privado. La sección protegida por middleware de autenticación (auth, verified) contiene el dashboard del usuario, gestión completa de planos mediante resource controller (crear, editar, eliminar, listar), historial de compras con acceso de solo lectura, sistema de invitaciones para que usuarios existentes puedan invitar a otros, y administración completa del perfil de usuario incluyendo edición y eliminación de cuenta. El sistema utiliza resource controllers para operaciones CRUD estándar, manteniendo convenciones RESTful, y aplica el principio de separación de responsabilidades donde cada controlador maneja un dominio específico de la aplicación

Guardado de documentos

Cuando un cliente inserta un plano, cada plano se deberá de estar guardando en la carpeta storage/app/public/blueprints o avatars si el usuario cambia de imagen, para evitar errores deberás de utilizar el comando `php artisan storage:link` esto con el fin de crear el acceso de la carpeta public y que no nos vaya a enviar el problema de 403 forbidden

Figura 38 Almacenamiento de Documentos en la plataforma



Fuente: Por los autores

4. Requerimientos Del Software

Requisito	Detalle
Sistema Operativo	Windows 10, Superior o Equivalentes
Procesador	Intel Core i3, superior o Equivalentes
Memoria RAM	8 GB o tamaños superiores
Disco Duro	100 GB o mayor capacidad
Resolución De Pantalla	1280 x 720 Pixeles
Periféricos	Teclado, Ratón Bocinas (Opcional)

Bibliografía

Aprendiendo a usar GitHub — Conociendo GitHub 0.1 documentation. (n.d.). Readthedocs.Io. Retrieved July 30, 2025, from <https://conociendogithub.readthedocs.io/en/latest/data/dinamica-de-uso/>

Composer. (s/f). Getcomposer.org. Recuperado el 30 de julio de 2025, de <https://getcomposer.org/>

Download Visual Studio Code. (s/f). Visualstudio.com. Recuperado el 30 de julio de 2025, de <https://code.visualstudio.com/download>

LeoKhoa. (s/f). *Laragon*. Laragon. Recuperado el 30 de julio de 2025, de <https://laragon.org/>

¿Qué es Visual Studio Code y cuáles son sus ventajas? (n.d.). Arsys. Retrieved July 30, 2025, from <https://www.arsys.es/blog/que-es-visual-studio-code-y-cuales-son-sus-ventajas>

(S/f). Php.net. Recuperado el 30 de julio de 2025, de <https://www.php.net/downloads.php>