



INTEGRANTES:

Edgar Adolfo Mejía Enamorado	202210050029
Elgin Said Maldonado Madrid	201720010317
Jose Luis Meraz Guerra	201810120012
Josue David Mora Bonilla	201910070159
Luis Enrique Cruz Gámez	202210010814
Sindy Osiris Zepeda Zelaya	202120010035

ASIGNATURA:

Movil I

CATEDRATICO:

Ricardo Enrique Iagos Mendoza

FECHA:

08/08/2024

# CONTENIDO

## Contenido

CONTENIDO.....	2
Introducción .....	3
Análisis del Problema o Proyecto .....	4
DESARROLLO .....	6
LIBRERIAS .....	6
DISEÑO .....	8
CODIGO DOCUMENTACION.....	10
MIGRACIONES .....	25
Views .....	26
ROUTES.....	31
Planificación de actividades (Diagrama de Gantt).....	37
Conclusiones Técnicas .....	39
BIBLIOGRAFIA.....	40
ANEXOS .....	41

## Introducción

En el contexto de la clase Movil 1 se nos ha encomendado la tarea de desarrollar una aplicación móvil que proporcione una solución integral para la gestión de pedidos en un restaurante, el proyecto que se presenta a continuación se titula “Aplicación Móvil para la Gestión de Pedidos para el Restaurante El Provenir Steaks”.

El Provenir Steaks es un restaurante especializado en ofrecer cortes de carne de alta calidad, y uno de los desafíos a los que se enfrenta es la optimización del proceso de gestión de pedidos para mejorar la eficiencia operativa y la experiencia del cliente la solución propuesta mediante esta aplicación móvil tiene como objetivo abordar este desafío proporcionando una aplicación intuitiva y eficiente que facilite la creación, seguimiento y gestión de pedidos tanto para los clientes como para el personal del restaurante.

# Análisis del Problema o Proyecto

## **Justificación del Proyecto**

El sector restaurantero se encuentra en un proceso continuo de innovación y adaptación tecnológica para satisfacer las crecientes expectativas de los clientes y optimizar sus operaciones internas, el restaurante El Provenir Steaks no es una excepción a esta tendencia actualmente la gestión de pedidos en el restaurante enfrenta varios desafíos, como la demora en la toma y procesamiento de órdenes, errores humanos en la transcripción de pedidos, y la falta de una comunicación eficiente entre el personal de servicio y la cocina estos problemas no solo afectan la eficiencia operativa del restaurante, sino que también impactan negativamente en la experiencia del cliente.

La implementación de una aplicación móvil para la gestión de pedidos ofrece una solución tecnológica que puede abordar estos problemas de manera efectiva la aplicación permitirá a los clientes realizar pedidos directamente desde sus dispositivos móviles, eliminando la necesidad de intermediarios y reduciendo el margen de error. Además, proporcionará al personal del restaurante una herramienta eficiente para gestionar y rastrear los pedidos en tiempo real, mejorando la coordinación y la velocidad del servicio.

## **Alcance del Proyecto**

El alcance del proyecto incluye el desarrollo e implementación de una aplicación móvil para la plataforma Android que cumpla con los siguientes objetivos y características:

1. Login de Usuarios: Autenticación segura para clientes y personal del restaurante.
2. Catálogo de Productos: Visualización detallada del menú con imágenes, descripciones y precios.
3. Creación de Pedidos: Interfaz intuitiva para que los clientes puedan seleccionar y personalizar sus pedidos.
4. Localización en Tiempo Real: Seguimiento del estado de los pedidos desde la cocina hasta la entrega.
5. Valoraciones: Funcionalidad para que los clientes puedan calificar y dejar comentarios sobre su experiencia.

La aplicación estará integrada con un sistema de gestión backend basado en VPS DigitalOcean y una REST API desarrollada en PHP Laravel. La base de datos utilizada será MySQL.

### **Arquitectura del Proyecto**

La arquitectura del proyecto se ha diseñado para ser escalable, segura y eficiente, y se compone de los siguientes componentes principales:

1. Cliente Móvil (Android): Desarrollado en Android Studio, este componente incluye todas las interfaces de usuario y la lógica de la aplicación necesaria para la interacción con los usuarios finales.
2. Servidor Backend: Implementado en una VPS utilizando tecnologías como PHP Laravel. Este servidor gestionará la lógica del negocio, autenticación de usuarios, procesamiento de pedidos y comunicación con la base de datos.
3. Base de Datos: Se utilizarán Firebase para autenticación y almacenamiento de datos en tiempo real, o MySQL/SQL Server para almacenamiento estructurado y consultas complejas.
4. REST API: Una REST API facilitará la comunicación entre el cliente móvil y el servidor backend, manejando las solicitudes HTTP y respondiendo con datos en formato JSON.
5. Seguridad: Implementación de medidas de seguridad para proteger los datos de los usuarios y asegurar las comunicaciones entre el cliente y el servidor mediante HTTPS y autenticación de usuarios.

Esta arquitectura modular permite una fácil escalabilidad y mantenimiento, asegurando que la aplicación pueda evolucionar y adaptarse a futuras necesidades del restaurante El Provenir Steaks.

# DESARROLLO

## **Aplicaciones para Desarrollar**

### 1. Aplicación Móvil para Clientes (Android)

- Funcionalidades Principales:
- Registro y Login: Permitir a los usuarios registrarse y autenticarse de forma segura.
- Exploración del Menú: Visualizar el catálogo de productos con imágenes, descripciones y precios.
- Creación y Personalización de Pedidos: Seleccionar productos, añadir especificaciones y realizar pedidos.
- Rastreo de Pedidos: Monitorear el estado del pedido en tiempo real desde la confirmación hasta la entrega.
- Valoraciones: Permitir a los usuarios calificar su experiencia.

### 2. Página web para el administrador del Restaurante

- Funcionalidades Principales:
- Login para el Personal: Autenticación segura para los empleados.
- Gestión de Pedidos: Visualizar y gestionar los pedidos entrantes.
- Actualización del Estado del Pedido: Actualizar el estado de los pedidos (en preparación, listo, entregado).
- Notificaciones Internas: Recibir notificaciones sobre nuevos pedidos y cambios en el estado de los pedidos.

# LIBRERIAS

## **Librerías clave y su función:**

Jetpack Compose: Ideal para crear interfaces de usuario modernas y declarativas. Simplifica la creación de layouts complejos y reactivos.

Material Components: Proporciona componentes de interfaz de usuario siguiendo las pautas de Material Design, asegurando una apariencia consistente y atractiva.

Retrofit: Facilita la creación de llamadas a APIs REST para gestionar la autenticación, la obtención de datos del menú, la creación de pedidos y el seguimiento de su estado.

OkHttp: Una librería HTTP cliente que complementa a Retrofit, ofreciendo características avanzadas como interceptores, caché y manejo de errores.

### **Base de datos local:**

Room: Una ORM (Object Relational Mapper) que permite interactuar con una base de datos SQLite de forma sencilla y segura. Ideal para almacenar datos locales del usuario, como su perfil y historial de pedidos.

### **Geolocalización:**

Fused Location Provider: Proporciona una API unificada para acceder a los servicios de ubicación del dispositivo, incluyendo GPS y redes móviles.

# DISEÑO

## LOGIN

El Porvenir Steaks

Login

Ingrese un correo

Ingrese una contraseña

ENTRAR

Regístrate ahora

## REGISTRO

El Porvenir Steaks

Registro

Ingrese su nombre

Ingrese su correo

Ingrese su contraseña

ENVIAR

Iniciar Sesión

## PERFIL

11:43 44%

El Porvenir Steaks

Perfil

Ingrese su nombre  
EnriqueL

Ingrese su correo  
enrique@gmail.com

Ingrese su contraseña

ACTUALIZAR


CAMBIAR CONTRASEÑA

## MENU

El Porvenir Steaks

MENÚ

DESCUENTO 40%




Parrilla Familiar - \$450

la creación de un nuevo plato para el menú del restaurante Parrilla Familiar. El plato se llama "Parrillada Familiar" y es una selección de las carnes más jugosas y sabrosas del restaurante, asadas a la perfección sobre las brasas. La parrillada incluye costillas de cerdo, chorizo argentino, picaña, pollo y una variedad de acompañamientos.

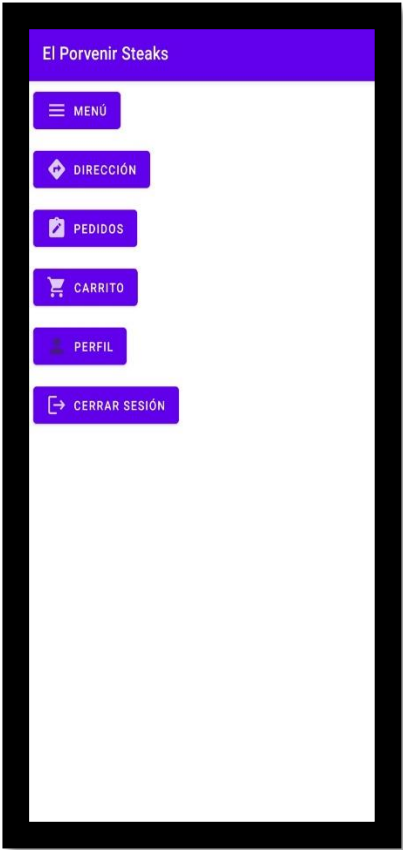
+

-

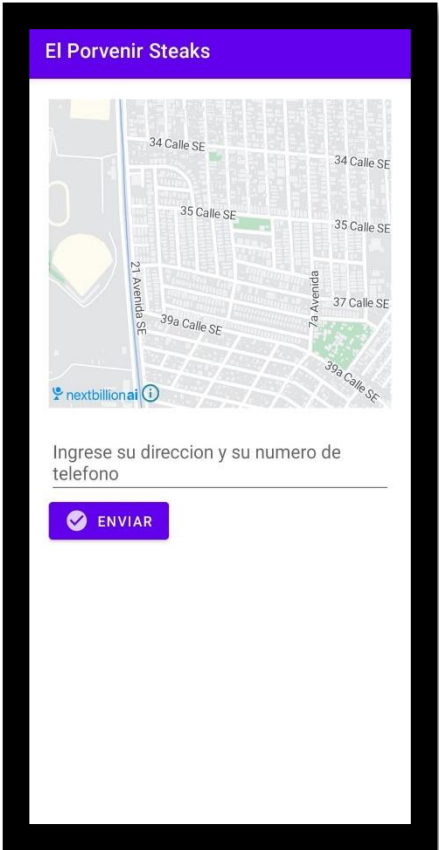




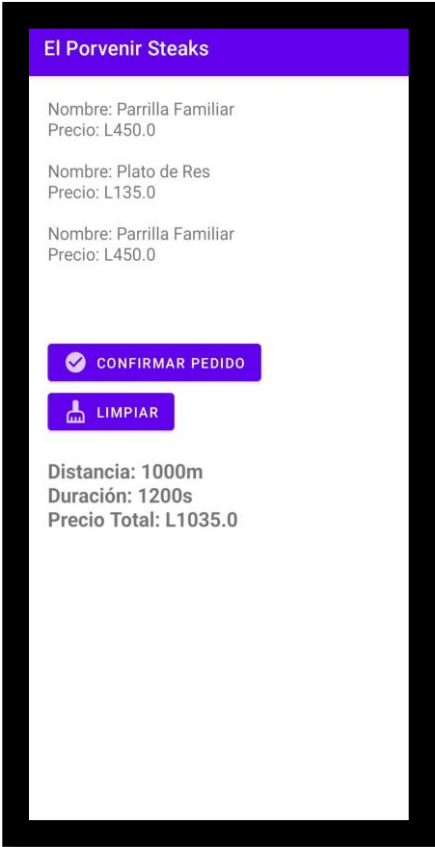
BARRA LATERAL



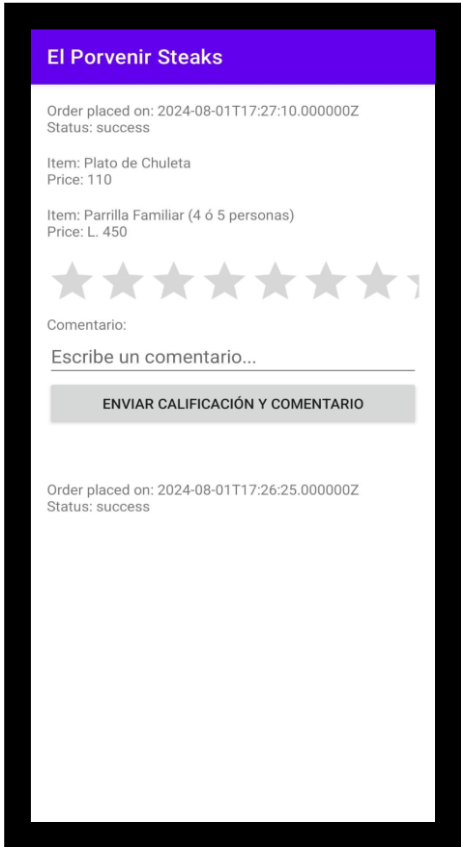
ENVIO PEDIDO



CONFIRMACION DE PEDIDO



VALIDACION DEL PEDIDO



## CODIGO DOCUMENTACION

Este documento presenta una propuesta detallada para el desarrollo de un sistema de pedidos de restaurante que combine la potencia de Android Studio para la aplicación móvil y Laravel para el backend la solución propuesta busca optimizar la gestión de pedidos, mejorar la experiencia del cliente y brindar una herramienta eficiente para los administradores del restaurante.

Http:

Controllers: Este directorio es donde se ubican los controladores que manejan las solicitudes HTTP entrantes y generan las respuestas los nombres de los archivos sugieren que esta aplicación maneja:

AuthApiManager.php: Probablemente se encarga de la autenticación a través de una API, como el inicio de sesión y el registro de usuarios.

```
namespace App\Http\Controllers;

use App\Models\User;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\Hash;

class AuthApiManager extends Controller
{
    // Método para el inicio de sesión de los usuarios
    function login(Request $request)
    {
        // Verifica si los campos de email y password están vacíos
        if (empty($request->email) && empty($request->password)) {
            return array("status" => "failed", "message" => "All fields are required");
        }

        // Busca en la base de datos un usuario con el email proporcionado
        $user = User::where("email", $request->email)->first();
```

```

if (!$user) {

    // Retorna un mensaje indicando que las credenciales son incorrectas
    return array("status" => "failed", "message" => "Retry with correct credentials");
}

// Obtiene las credenciales del request
$credentials = $request->only("email", "password");

// Intenta autenticar al usuario utilizando las credenciales
if (Auth::attempt($credentials)) {

    // Retorna un mensaje de éxito con los datos del usuario
    return array("status" => "success",
        "message" => "Login successful",
        "name" => $user->name, "email" => $user->email);
}

// Retorna un mensaje indicando que las credenciales son incorrectas
return array("status" => "failed", "message" => "Retry with correct credentials");
}

// Método para el registro de nuevos usuarios
function registration(Request $request)
{
    // Verifica si los campos de nombre, email y password están vacíos
    if (empty($request->name) && empty($request->email) && empty($request->password)) {
        return "failed";
    }

    // Crea un nuevo usuario en la base de datos
    $user = User::create([
        'type' => "customer", // Establece el tipo de usuario como "customer"
        "name" => $request->name, // Asigna el nombre del usuario
        "email" => $request->email, // Asigna el email del usuario
    ]);
}

```

```

        "password" => Hash::make($request->password) // Hashea la contraseña del usuario
    });

    if(!$user){
        // Retorna "failed" si la creación del usuario falla
        return "failed";
    }

    // Retorna "success" si la creación del usuario es exitosa
    return "success";
}
}

```

**AuthManager.php:** Maneja la lógica de autenticación general, como la verificación de credenciales.

```

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\Session;

class AuthManager extends Controller
{
    // Método para mostrar la vista de inicio de sesión
    function login(){
        return view("login");
    }

    // Método para manejar el envío del formulario de inicio de sesión
    function loginPost(Request $request){
        // Validación de los campos del formulario
        $request->validate([
            'email' => 'required|email', // El campo email es obligatorio y debe ser una dirección de correo electrónico válida
            'password' => 'required' // El campo password es obligatorio

```

```

    });

    // Obtiene las credenciales del request
    $credentials = $request->only("email", "password");

    // Intenta autenticar al usuario utilizando las credenciales
    if (Auth::attempt($credentials)) {
        // Redirige al usuario a la página del dashboard con un mensaje de éxito
        return redirect()->intended(route("dashboard"))->with("success", "Login success");
    }

    // Redirige al usuario a la página de inicio de sesión con un mensaje de error
    return redirect()->intended(route("login"))->with("error", "Login success");
}

// Método para cerrar la sesión del usuario
function logout(){
    // Limpia todas las sesiones activas
    Session::flush();

    // Desconecta al usuario
    Auth::logout();

    // Redirige al usuario a la página de inicio de sesión
    return redirect(route("login"));
}
}

```

**Controller.php:** Controla la base o contener lógica común a otros controladores.

**CustomerManager.php:** Gestiona las operaciones relacionadas con los clientes, como la creación, actualización y eliminación de cuentas de clientes.

```
namespace App\Http\Controllers;
```

```

use App\Models\User;

use Illuminate\Http\Request;

class CustomerManager extends Controller
{
    // Método para actualizar la dirección del usuario
    function updateAddress(Request $request)
    {
        // Busca en la base de datos un usuario con el email proporcionado
        $user = User::where("email", $request->email)->first();

        // Asigna los nuevos valores de dirección, latitud y longitud
        $user->destination_address = $request->address;
        $user->destination_lat = $request->latitude;
        $user->destination_lon = $request->longitude;

        // Guarda los cambios en la base de datos
        if ($user->save()) {
            // Retorna "success" si la actualización es exitosa
            return "success";
        }

        // Retorna "failed" si la actualización falla
        return "failed";
    }
}

```

**DeliveryBoyManager.php:** Maneja las operaciones relacionadas con los repartidores, como la asignación de pedidos.

```

namespace App\Http\Controllers;

use App\Models\Orders;

use Illuminate\Http\Request;

```

```

class DeliveryBoyManager extends Controller
{
    // Método para obtener las entregas asignadas a un repartidor
    function getDelivery(Request $request)
    {
        // Busca en la base de datos las órdenes asignadas al repartidor con el email proporcionado
        $delivery = Orders::where("delivery_boy_email", $request->email)
            ->where("status", "assigned") // Filtra por estado "assigned" (asignado)
            ->orderBy("id", "DESC") // Ordena las órdenes por id en orden descendente
            ->get(); // Obtiene los resultados

        // Retorna las órdenes encontradas
        return $delivery;
    }

    // Método para actualizar el estado de una orden
    function markStatus(Request $request, $status)
    {
        // Busca en la base de datos la orden con el id proporcionado
        $order = Orders::where("id", $request->order_id)->first();

        // Actualiza el estado de la orden
        $order->status = $status;

        // Guarda los cambios en la base de datos
        if ($order->save()) {
            // Retorna "success" si la actualización es exitosa
            return "success";
        }

        // Retorna "failed" si la actualización falla
        return "failed";
    }
}

```

```

// Método para marcar una orden como exitosa
function markStatusSuccess(Request $request)
{
    // Llama a markStatus con el estado "success"
    return $this->markStatus($request, "success");
}

// Método para marcar una orden como fallida
function markStatusFailed(Request $request)
{
    // Llama a markStatus con el estado "failed"
    return $this->markStatus($request, "failed");
}
}

```

**OrderManager.php:** Gestiona las operaciones relacionadas con los pedidos, como la creación, actualización y seguimiento de pedidos.

```

namespace App\Http\Controllers;

use App\Models\Cart;
use App\Models\Orders;
use App\Models\Products;
use App\Models\User;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\DB;
use Illuminate\Support\Facades\Http;

class OrderManager extends Controller
{
    // Método para obtener nuevas órdenes con estado "open" (abierto)
    function newOrders()
    {
        $orders = Orders::where("status", "open")->get();
    }
}

```



```

$orders = json_decode(json_encode($orders));

$delivery_boys = User::where("type", "delivery")->get();

$products = Products::get();

// Asigna los detalles del producto a cada orden
foreach ($orders as $key => $order) {
    $order_item_ids = json_decode($order->items);
    foreach ($order_item_ids as $key2 => $order_item) {
        foreach ($products as $product) {
            if ($order_item->item_id == $product->id) {
                $orders[$key]->item_details[$key2] = $product;
            }
        }
    }
}

// Retorna la vista "dashboard" con las órdenes y los repartidores
return view("dashboard", compact("orders", "delivery_boys"));
}

// Método para asignar una orden a un repartidor
function assignOrder(Request $request)
{
    $order = Orders::where("id", $request->order_id)->first();
    $order->delivery_boy_email = $request->delivery_boy_email;
    $order->status = "assigned";

    // Guarda los cambios y redirige con un mensaje de éxito o error
    if ($order->save()) {
        return redirect(route("dashboard"))
            ->with("success", "Order assigned successfully");
    }
    return redirect(route("dashboard"))

```

```

        ->with("error", "Failed to assign Order");
    }

// Método para listar todas las órdenes
function listOrders()
{
    $orders = Orders::orderBy("id", "DESC")->get();
    $orders = json_decode(json_encode($orders));
    $products = Products::get();

    // Asigna los detalles del producto a cada orden
    foreach ($orders as $key => $order) {
        $order_item_ids = json_decode($order->items);
        foreach ($order_item_ids as $key2 => $order_item) {
            foreach ($products as $product) {
                if ($order_item->item_id == $product->id) {
                    $orders[$key]->item_details[$key2] = $product;
                }
            }
        }
    }

    // Retorna la vista "order" con las órdenes
    return view("order", compact("orders"));
}

// Método para agregar un producto al carrito
function addToCart(Request $request)
{
    $cart = new Cart();
    $cart->item_id = $request->item_id;
    $cart->user_email = $request->user_email;

```

```

// Guarda el carrito y retorna "success" o "failed"
if ($cart->save()) {
    return "success";
}
return "failed";
}

// Método para eliminar un producto del carrito
function removeFromCart(Request $request)
{
    $cart = Cart::where("item_id", $request->item_id)
        ->where("user_email", $request->user_email)->first();
    if ($cart == null) {
        return "failed";
    }
    if ($cart->delete()) {
        return "success";
    }

    return "failed";
}

// Método para obtener el contenido del carrito de un usuario
function getCart(Request $request)
{
    $item_id = array();
    $count_items = DB::select(
        "SELECT item_id, COUNT(item_id) as num_item from cart
        where user_email = '" . $request->user_email . "'
        GROUP BY item_id");
    foreach ($count_items as $key => $item) {
        $item_id[$key] = $item->item_id;
    }
}

```

```

$user = User::where("email", $request->user_email)->first();

$dis_dur = $this->calculateEstimatedTime($user);

$products = Products::whereIn('id', $item_id)->get();

foreach ($count_items as $item) {

    foreach ($products as $key => $product) {

        if ($item->item_id == $product->id) {

            $products[$key]->numItem = $item->num_item;

        }

    }

}

$data = array();

array_push($data, array("cart" => json_decode($products),

    "duration" => $dis_dur['duration'], "distance" => $dis_dur['distance']));

return $data;

}

// Método para confirmar el contenido del carrito y crear una orden

function confirmCart(Request $request)

{

    $cart = Cart::select("item_id")->where("user_email", $request->user_email)->get();

    if (empty($cart->first())) {

        return "failed";

    }

    $user = User::where("email", $request->user_email)->first();

    $order = new Orders();

    $order->customer_email = $request->user_email;

    $order->items = $cart;

    $order->status = "open";

    $order->destination_address = $user->destination_address;

    $order->destination_lat = $user->destination_lat;

    $order->destination_lon = $user->destination_lon;

    // Guarda la orden y limpia el carrito si la orden se guarda correctamente

```

```

        if ($order->save()) {
            if ($this->clearCart($request) == "success") {
                return "success";
            }
        }

        return "failed";
    }

    // Método para limpiar el contenido del carrito de un usuario
    function clearCart(Request $request)
    {
        if (Cart::where("user_email", $request->user_email)->delete()) {
            return "success";
        }

        return "failed";
    }

    // Método para calcular el tiempo estimado de entrega usando una API externa
    function calculateEstimatedTime($user)
    {
        $origin_lat = 34.0581903; // Latitud de origen (Los Angeles - Ubicación de Starbucks)
        $origin_lon = -118.2383913; // Longitud de origen

        $apiURL = "https://api.nextbillion.io/distancematrix/json?origins=$origin_lat,$origin_lon&destinations=$user->destination_lat,$user->destination_lon&mode=4w&key=your-nextbillion-api-key-here"; // URL de la API de nextbillion.ai

        $response = json_decode(Http::get($apiURL));

        $dist_dur['distance'] = $response->rows[0]->elements[0]->distance->value;
        $dist_dur['duration'] = $response->rows[0]->elements[0]->duration->value;

        return $dist_dur;
    }

```

```

    }

    // Método para obtener las órdenes de un usuario específico
    function getOrders(Request $request){
        $orders = Orders::where("customer_email", $request->email)->orderBy("id", "DESC")->get();
        $orders = json_decode(json_encode($orders));
        $products = Products::get();
        foreach ($orders as $key => $order) {
            $order_item_ids = json_decode($order->items);
            foreach ($order_item_ids as $key2 => $order_item) {
                foreach ($products as $product) {
                    if ($order_item->item_id == $product->id) {
                        $orders[$key]->item_details[$key2] = $product;
                    }
                }
            }
        }
    }

    return $orders;
}
}

```

**ProductManager.php:** Gestiona las operaciones relacionadas con los productos, como la gestión del catálogo de productos.

```

namespace App\Http\Controllers;

use App\Models\Products;
use Illuminate\Http\Request;

class ProductManager extends Controller
{
    // Método para obtener todos los productos
    function getProducts(){
        return Products::get(); // Retorna todos los productos de la base de datos
    }
}

```

```

}

// Método para listar todos los productos en la vista "products"
function listProducts(){
    $products = $this->getProducts(); // Obtiene todos los productos
    return view("products", compact("products")); // Retorna la vista "products" con los productos
}

// Método para agregar un nuevo producto
function addProducts(Request $request){
    $product = new Products();
    $product->name = $request->name; // Asigna el nombre del producto
    $product->description = $request->description; // Asigna la descripción del producto
    $product->price = $request->price; // Asigna el precio del producto
    $product->image = $request->image; // Asigna la imagen del producto

    // Guarda el producto y redirige con un mensaje de éxito o error
    if($product->save()){
        return redirect(route("products"))
            ->with("success", "Correcto, se agregó un nuevo plato al menú");
    }
    return redirect(route("products"))
        ->with("error", "Problema, no se pudo ingresar el platillo");
}

// Método para eliminar un producto
function deleteProducts(Request $request){
    // Elimina el producto con el ID dado y redirige con un mensaje de éxito o error
    if(Products::where("id",$request->id)->delete()){
        return redirect(route("products"))
            ->with("success", "Correcto, se eliminó un plato del menú");
    }
    return redirect(route("products"))

```

```
->with("error", "Problema, no se pudo eliminar el platillo");  
}
```

### **Models:**

Contiene las clases que representan los modelos de datos de la aplicación. Cada archivo PHP representa una tabla en la base de datos. Por ejemplo:

Cart.php: Representa los carritos de compra de los usuarios.

Orders.php: Representa los pedidos realizados.

Products.php: Representa los productos disponibles para la venta.

User.php: Representa a los usuarios de la aplicación. Models:

Contiene las clases que representan los modelos de datos de la aplicación. Cada archivo PHP representa una tabla en la base de datos. Por ejemplo:

Cart.php: Representa los carritos de compra de los usuarios.

Orders.php: Representa los pedidos realizados.

Products.php: Representa los productos disponibles para la venta.

User.php: Representa a los usuarios de la aplicación.

**(Solamente se asignará un ejemplo ya que es la misma asignación en las demás paginas php.)**

```
namespace App\Models;
```

```
use Illuminate\Database\Eloquent\Factories\HasFactory;
```

```
use Illuminate\Database\Eloquent\Model;
```

```
class Orders extends Model
```

```
{
```

```
    // Define la tabla asociada a este modelo
```

```
    protected $table = "orders";
```

```
    // Usa el trait HasFactory para habilitar las factorías en este modelo
```

```
    use HasFactory;
```

```
}
```



## MIGRACIONES

2014\_10\_12\_000000\_create\_users\_table.php:

Se creó la tabla users, que es esencial para cualquier aplicación que requiera autenticación de usuarios. Esta tabla típicamente contiene campos como id, name, email, password y otros datos relevantes para los usuarios.

2014\_10\_12\_100000\_create\_password\_resets\_table.php:

Se creó la tabla password\_resets. Esta tabla se utiliza para almacenar tokens de restablecimiento de contraseña cuando un usuario olvida su contraseña.

2019\_08\_19\_000000\_create\_failed\_jobs\_table.php:

Se creó la tabla failed\_jobs. Esta tabla se utiliza para almacenar información sobre trabajos en cola que fallaron en su ejecución.

2019\_12\_14\_000001\_create\_personal\_access\_tokens\_table.php:

Se creó la tabla personal\_access\_tokens. Esta tabla se utiliza para almacenar tokens de acceso personales, que pueden ser utilizados por aplicaciones de terceros para acceder a la aplicación en nombre del usuario.

2024\_07\_15\_072325\_products.php, 2024\_07\_15\_052650\_cart.php,  
2024\_07\_15\_052852\_orders.php:

Estas migraciones son específicas de la aplicación. Basándose en sus nombres, podemos inferir que crearon las tablas products, cart y orders, respectivamente. Estas tablas probablemente se utilizan para gestionar los productos, los carritos de compra y los pedidos en tu aplicación de entrega de comida.

**(Solamente se asignará un ejemplo ya que es la misma asignación en las demás paginas php.)**

```
use Illuminate\Database\Migrations\Migration;
```

```
use Illuminate\Database\Schema\Blueprint;
```

```
use Illuminate\Support\Facades\Schema;
```

```
return new class extends Migration
```

```
{
```

```
    /**
```

```
     * Ejecutar las migraciones.
```

```
     *
```

```
     * @return void
```

```
    */
```

```

public function up()
{
    // Crear la tabla 'products' en la base de datos
    Schema::create('products', function (Blueprint $table) {
        // Definir las columnas de la tabla
        $table->id(); // Columna ID autoincremental y clave primaria
        $table->string('name'); // Columna 'name' de tipo cadena
        $table->string('description', 1000); // Columna 'description' de tipo cadena con un límite de 1000
        caracteres
        $table->string('price'); // Columna 'price' de tipo cadena
        $table->string('image'); // Columna 'image' de tipo cadena
        $table->timestamps(); // Columnas 'created_at' y 'updated_at' para gestionar las marcas de
        tiempo
    });
}

/*
 *
 * Revertir las migraciones.
 ** @return void
 */
public function down()
{
    // Eliminar la tabla 'products' si existe
    Schema::dropIfExists('products');
}
};

```

## Views

Este es el directorio principal donde se almacenan todas las vistas de tu aplicación.

include: Dentro de este directorio, es probable que encuentres vistas más pequeñas y reutilizables que se incluyen en otras vistas más grandes.

footer.blade.php, header.blade.php: Estos archivos contienen el código HTML para el pie y el encabezado de tus páginas, respectivamente. Son elementos comunes que se repiten en muchas páginas y, por lo tanto, se suelen separar en archivos individuales para una mejor organización y mantenimiento.

dashboard.blade.php: Probablemente muestre el panel de control del usuario, con información personalizada y opciones.

```
@extends('layout')
```

```
@section('content')
```

```
<main class="container my-5" style="max-width: 700px">
```

```
<div>
```

```
<ul class="list-group">
```

```
<!-- Iterar sobre la colección de pedidos -->
```

```
@forelse($orders as $order)
```

```
<li class="list-group-item">
```

```
<div class="row">
```

```
<!-- Detalles del pedido -->
```

```
<div class="col-6">
```

```
<div class="fw-bold">
```

```
<!-- Mostrar los detalles de cada ítem en el pedido -->
```

```
@foreach($order->item_details as $item_details)
```

```
<div>{{ $item_details->name }}</div>
```

```
@endforeach
```

```
<br>
```

```
<!-- Mostrar la dirección de destino del pedido -->
```

```
Address: {{ $order->destination_address }}
```

```
</div>
```

```
</div>
```

```
<!-- Formulario para asignar un repartidor al pedido -->
```

```
<div class="col-6">
```

```
<form action="{{ route('order.assign') }}" method="POST">
```

```
@csrf
```

```
<!-- Campo oculto para el ID del pedido -->
```

```
<div class="form-floating">
```

```
<input name="order_id" value="{{ $order->id }}" hidden>
```

```

        <!-- Selector para elegir el repartidor -->

        <select class="form-select" id="delivery_boy_email" name="delivery_boy_email"
aria-label="Floating label select example">

            @foreach($delivery_boys as $delivery_boy)

                <option

                    value="{{ $delivery_boy->email }}">{{ $delivery_boy->name }}</option>

                @endforeach

            </select>

            <label for="delivery_boy_email">Seleccionar delivery</label>

        </div>

        <!-- Botón para enviar el formulario -->

        <input type="submit" class="btn btn-success rounded-pill" value="Assign">

    </form>

</div>

</div>

</li>

@empty

    <!-- Mensaje cuando no hay pedidos nuevos -->

    <li class="list-group-item">

        <div class="alert alert-warning">No hay pedidos nuevos</div>

    </li>

@endforeach

</ul>

</div>

</main>

@endsection

```

login.blade.php: Contiene el formulario de inicio de sesión.

order.blade.php: Muestra información sobre los pedidos realizados.

products.blade.php: Muestra una lista de productos disponibles.

```
@extends('layout')
```

```
@section('content')
```

```

    <main class="container my-5" style="max-width: 900px">

        <div class="row">

```

```

<!-- Mostrar mensajes de error -->

@if ($errors->any())
    <div class="col-12">
        @foreach ($errors->all() as $error)
            <div class="alert alert-danger">{{$error}}</div>
        @endforeach
    </div>
@endif

<!-- Mostrar mensaje de éxito si existe -->
@if(session()->has('success'))
    <div class="alert alert-success alert-dismissible">
        {{session('success')}}
    </div>
@endif

<!-- Mostrar mensaje de error si existe -->
@if(session()->has('error'))
    <div class="alert alert-danger alert-dismissible">
        {{session('error')}}
    </div>
@endif

<!-- Formulario para agregar un nuevo plato -->
<div class="col-6 col-md-6">
    <div class="fs-5 fw-bold mb-2">Agregar nuevo platillo:</div>
    <form method="POST" action="{{route('product.add')}}">
        @csrf
        <!-- Campo para el nombre del plato -->
        <div class="mb-3">
            <label class="form-label">Nombre del Plato</label>
            <input type="text" name="name" class="form-control">
        </div>
    </form>
</div>

```

```

<!-- Campo para la descripción del plato -->
<div class="mb-3">
  <label class="form-label">Descripción del plato</label>
  <input type="text" name="description" class="form-control">
</div>

<!-- Campo para el precio del plato -->
<div class="mb-3">
  <label class="form-label">Precio Lps</label>
  <input type="text" name="price" class="form-control">
</div>

<!-- Campo para la URL de la imagen del plato -->
<div class="mb-3">
  <label class="form-label">Imagen url</label>
  <input type="text" name="image" class="form-control">
</div>

<!-- Botón para enviar el formulario -->
<button type="submit" class="btn btn-primary">Agregar</button>
</form>
</div>

<!-- Listado de platos existentes -->
<div class="col-12 col-md-6">
  <div class="fs-5 fw-bold mb-2 text-decoration-underline">Menú de El Porvenir Steaks:</div>
  <ul class="list-group">
    <!-- Iterar sobre la colección de productos -->
    @foreach($products as $product)
      <li class="list-group-item">
        <div class="row">
          <!-- Información del producto -->
          <div class="col-8">
            <div class="fw-bold">{{ $product->name }} | Precio: Lps{{ $product->price }}</div>
            <small>{{ $product->description }}</small>
          </div>

```

```

        <!-- Imagen del producto y enlace para eliminar -->

        <div class="col-4">

            <a href="{{route('product.delete')}}?id={{${product->id}}}">Delete</a>

        </div>

    </div>

</li>

@endforeach

</ul>

</div>

</div>

</main>

@endsection

```

layout.blade.php: Este archivo suele ser una plantilla principal que define la estructura general de tus páginas. Define el layout (diseño) básico de la página, incluyendo elementos como el encabezado, el pie de página y las secciones principales del contenido. Otras vistas, como dashboard.blade.php, suelen extender esta plantilla layout.blade.php y reemplazar solo las secciones específicas que necesitan ser personalizadas.

## ROUTES

### routes/api.php

Propósito: Define las rutas para la API de tu aplicación.

Funcionalidad: Este archivo es ideal para definir endpoints que serán consumidos por aplicaciones externas, como aplicaciones móviles o frontend de una SPA (Single Page Application). Las rutas definidas aquí suelen ser más simples y directas, ya que están diseñadas para devolver datos en formato JSON.

```

<?php

use App\Http\Controllers\AuthApiManager;

use App\Http\Controllers\DeliveryBoyManager;

use App\Http\Controllers\OrderManager;

use App\Http\Controllers\ProductManager;

use Illuminate\Support\Facades\Route;

use App\Http\Controllers\CustomerManager;

```

// Rutas para autenticación de usuarios

```
Route::any("/users/login", [AuthApiManager::class, "login"]);
```

// Ruta para el inicio de sesión de usuarios, utiliza el método "login" del controlador AuthApiManager.

```
Route::any("/users/register", [AuthApiManager::class, "registration"]);
```

// Ruta para el registro de nuevos usuarios, utiliza el método "registration" del controlador AuthApiManager.

// Rutas para la gestión de repartidores (delivery boys)

```
Route::any("/users/delivery", [DeliveryBoyManager::class, "getDelivery"]);
```

// Ruta para obtener información sobre repartidores, utiliza el método "getDelivery" del controlador DeliveryBoyManager.

```
Route::any("/users/delivery/success", [DeliveryBoyManager::class, "markStatusSuccess"]);
```

// Ruta para marcar un repartidor como exitoso, utiliza el método "markStatusSuccess" del controlador DeliveryBoyManager.

```
Route::any("/users/delivery/failed", [DeliveryBoyManager::class, "markStatusFailed"]);
```

// Ruta para marcar un repartidor como fallido, utiliza el método "markStatusFailed" del controlador DeliveryBoyManager.

// Ruta para la gestión de productos

```
Route::any("/product/list", [ProductManager::class, "getProducts"]);
```

// Ruta para obtener la lista de productos, utiliza el método "getProducts" del controlador ProductManager.

// Rutas para la gestión del carrito de compras

```
Route::any("/users/cart/add", [OrderManager::class, "addToCart"]);
```

// Ruta para agregar un producto al carrito, utiliza el método "addToCart" del controlador OrderManager.

```
Route::any("/users/cart/remove", [OrderManager::class, "removeFromCart"]);
```

// Ruta para eliminar un producto del carrito, utiliza el método "removeFromCart" del controlador OrderManager.



```
Route::any("/users/cart/list", [OrderManager::class, "getCart"]);
```

// Ruta para obtener la lista de productos en el carrito, utiliza el método "getCart" del controlador OrderManager.

```
Route::any("/users/cart/confirm", [OrderManager::class, "confirmCart"]);
```

// Ruta para confirmar los productos en el carrito, utiliza el método "confirmCart" del controlador OrderManager.

```
Route::any("/users/cart/clear", [OrderManager::class, "clearCart"]);
```

// Ruta para vaciar el carrito de compras, utiliza el método "clearCart" del controlador OrderManager.

```
Route::any("/users/orders/list", [OrderManager::class, "getOrders"]);
```

// Ruta para obtener la lista de pedidos, utiliza el método "getOrders" del controlador OrderManager.

// Ruta para actualizar la dirección del cliente

```
Route::any("/users/address/update", [CustomerManager::class, "updateAddress"]);
```

// Ruta para actualizar la dirección del cliente, utiliza el método "updateAddress" del controlador CustomerManager.

### **routes/channels.php**

Propósito: Define los canales de broadcast para la comunicación en tiempo real.

Funcionalidad: Este archivo es crucial para configurar los canales que se utilizarán para enviar eventos en tiempo real a los clientes conectados. Laravel utiliza Broadcast para facilitar la implementación de funcionalidades como chat en vivo, notificaciones en tiempo real, etc.

```
<?php
```

```
use Illuminate\Support\Facades\Broadcast;
```

```
/*
```

```
|-----
```

```
| Broadcast Channels
```

```
|-----
```

```
|
```

```
| Aquí puedes registrar todos los canales de transmisión de eventos que tu
```

```
| aplicación admite. Los callbacks de autorización de canales dados se
```

```
| utilizan para verificar si un usuario autenticado puede escuchar el canal.
|
*/
```

```
// Define un canal de transmisión para el modelo 'User'
// El nombre del canal incluye el identificador del usuario ({id})
Broadcast::channel('App.Models.User.{id}', function ($user, $id) {
    // La función de autorización verifica si el ID del usuario autenticado
    // coincide con el ID proporcionado en la solicitud del canal
    return (int) $user->id === (int) $id;
});
```

### **routes/console.php**

Propósito: Define comandos de consola personalizados.

Funcionalidad: Este archivo te permite crear comandos personalizados que se pueden ejecutar desde la línea de comandos. Estos comandos pueden automatizar tareas como generar datos de prueba, ejecutar tareas programadas o realizar cualquier otra acción que no requiera una interfaz web.

```
<?php
```

```
use Illuminate\Foundation\Inspiring;
use Illuminate\Support\Facades\Artisan;
```

```
/*
|-----
| Console Routes
|-----
|
| Este archivo es donde puedes definir todos tus comandos de consola basados en
| Cierre (Closure). Cada Cierre se enlaza con una instancia de comando, permitiendo
| un enfoque simple para interactuar con los métodos de entrada/salida de cada comando.
|
*/
```

```
// Define un comando de consola personalizado llamado 'inspire'
```

```

Artisan::command('inspire', function () {
    // Utiliza el método `comment` para mostrar una cita inspiradora
    // La cita se obtiene utilizando la clase `Inspiring` y su método `quote()`
    $this->comment(Inspiring::quote());
})->purpose('Display an inspiring quote');

```

### **routes/web.php**

es el corazón de tu aplicación Laravel, ya que define cómo los usuarios interactúan con tu sitio web. Al comprender cómo funciona, podrás crear aplicaciones web dinámicas y personalizadas.

```
<?php
```

```

use App\Http\Controllers\AuthManager;
use App\Http\Controllers\OrderManager;
use App\Http\Controllers\ProductManager;
use App\Http\Middleware\RoleAdmin;
use Illuminate\Support\Facades\Route;

// Ruta raíz que devuelve un saludo
Route::get('/', function () {
    return "hi";
});

// Rutas para autenticación
Route::get("login", [AuthManager::class, "login"]->name("login"));

// Ruta para cerrar sesión
Route::get("logout", [AuthManager::class, "logout"]->name("logout"));

// Ruta para procesar el formulario de inicio de sesión
Route::post("login", [AuthManager::class, "loginPost"]->name("login.post"));

// Rutas agrupadas bajo el prefijo 'admin' con el middleware RoleAdmin
Route::prefix("admin")->middleware(RoleAdmin::class)->group(function(){
    // Ruta para mostrar el dashboard de administración con nuevos pedidos
    Route::get('dashboard', [OrderManager::class, "newOrders"]->name('dashboard'));

```

```
// Ruta para mostrar la lista de productos
Route::get('products', [ProductManager::class, "listProducts"]->name("products"));

// Ruta para agregar un nuevo producto
Route::post('products', [ProductManager::class, "addProducts"]->name("product.add"));

// Ruta para eliminar un producto
Route::get('product/delete', [ProductManager::class, "deleteProducts"]->name("product.delete"));

// Ruta para asignar un pedido a un repartidor
Route::post('order/assign', [OrderManager::class, "assignOrder"]->name("order.assign"));

// Ruta para listar todos los pedidos
Route::get('order/list', [OrderManager::class, "listOrders"]->name("order.list"));

});
```

# Planificación de actividades (Diagrama de Gantt)

## **Semana 1 (5 de junio - 11 de junio): Planificación y Análisis**

- Definición de Requisitos (5-6 de junio)
- Reunión con integrantes del grupo
- Recolección de requisitos
- Análisis de viabilidad

## **Diseño del Proyecto (7-9 de junio)**

- Diseño de la arquitectura
- Diseño de la base de datos
- Prototipo de la interfaz de usuario

## **Semana 2 (12 de junio - 18 de junio): Configuración del Entorno y Desarrollo Inicial**

- Configuración del Entorno de Desarrollo (12-14 de junio)
- Configuración de Android Studio
- Configuración del servidor Clever cloud
- Configuración de MySQL

## **Desarrollo de la API REST (15-16 de junio)**

- Configuración inicial del proyecto en PHP Laravel
- Implementación de endpoints básicos (registro, login, obtención del menú)

## **Semana 3-5 (19 de junio - 9 de julio): Desarrollo de Funcionalidades Principales**

- Desarrollo de la Aplicación Móvil para Clientes (19 de junio - 2 de julio)
- Registro y Login de Usuarios (19-21 de junio)
- Exploración del Menú (22-24 de junio)
- Creación y Personalización de Pedidos (26-29 de junio)
- Rastreo de Pedidos

### Semana 6 (10 de julio - 16 de julio): Desarrollo de Funcionalidades para el Personal del Restaurante

- Desarrollo página web para el Personal del Restaurante (10-16 de julio)
- Login para el Personal (10-12 de julio)
- Gestión y Actualización del Estado de Pedidos (13-16 de julio)

### Semana 7 (17 de julio - 23 de julio): Integración y Pruebas

- Integración de Componentes (17-18 de julio)
- Integración de la aplicación móvil con la REST API
- Pruebas de integración
- Pruebas del Sistema (19-21 de julio)
- Pruebas funcionales y de rendimiento

### Semana 8 (24 de julio - 30 de julio): Despliegue y Documentación

- Despliegue del Sistema (24-25 de julio)
- Configuración del servidor Clever cloud
- Documentación del Proyecto (26-27 de julio)
- Documentación técnica

### Semana 9 (28 de julio - 30 de julio): Revisión Final y Presentación

- Presentacion (28-30 de julio)

(TECNOLOGIAS USADAS PARA ALOJAMIENTO EN LA NUBE)

- La vps está configurada con Ubuntu server
- Apache2 como servidor web

## Conclusiones Técnicas

- **Optimización de la Gestión de Pedidos:** La implementación de la aplicación móvil ha optimizado significativamente la gestión de pedidos en el restaurante "El Provenir Steaks" los clientes pueden realizar sus pedidos de manera rápida y eficiente, y el personal del restaurante puede gestionar y actualizar el estado de los pedidos en tiempo real, esto ha mejorado la eficiencia operativa y ha reducido errores en el procesamiento de pedidos.
- **Integración de Tecnologías Modernas:** Utilizando tecnologías modernas como Android Studio para el desarrollo de la aplicación móvil, PHP Laravel para la REST API, y bases de datos como MySQL, se ha asegurado una arquitectura robusta y escalable, estas herramientas han permitido crear una aplicación segura, rápida y fácil de mantener.
- **Mejora en la Experiencia del Usuario:** El diseño intuitivo de la interfaz de usuario y la implementación de funcionalidades clave como la exploración del menú, la personalización de pedidos, y el rastreo en tiempo real han mejorado la experiencia del usuario.
- **Facilidad de Uso para el Personal del Restaurante:** La aplicación móvil desarrollada para el personal del restaurante facilita el acceso a la información de los pedidos y permite actualizarlos rápidamente. Esto reduce el tiempo de respuesta y mejora la comunicación interna, lo que resulta en un servicio más rápido y eficiente para los clientes.
- **Despliegue y Mantenimiento Simplificados:** El despliegue de la REST API en un servidor VPS han sido procesos relativamente sencillos gracias a la planificación y configuración inicial del entorno de desarrollo la documentación técnica asegura que el sistema pueda ser mantenido y actualizado con facilidad en el futuro.
- **Seguridad y Protección de Datos:** La implementación de medidas de seguridad adecuadas, tanto en la aplicación móvil como en la REST API, ha garantizado la protección de los datos personales y financieros de los usuarios. Esto es crucial para mantener la confianza de los clientes y cumplir con las normativas de protección de datos.

## BIBLIOGRAFIA

[Funcionamiento de app de delivery básica en Android Studio con java y SQLite \(youtube.com\)](#)

<https://juice-studio.com/lista-de-las-15-mejores-librerias-de-android/>

[¡Cómo tener tu propia base de datos SQL en la nube y gratis! \(youtube.com\)](#)

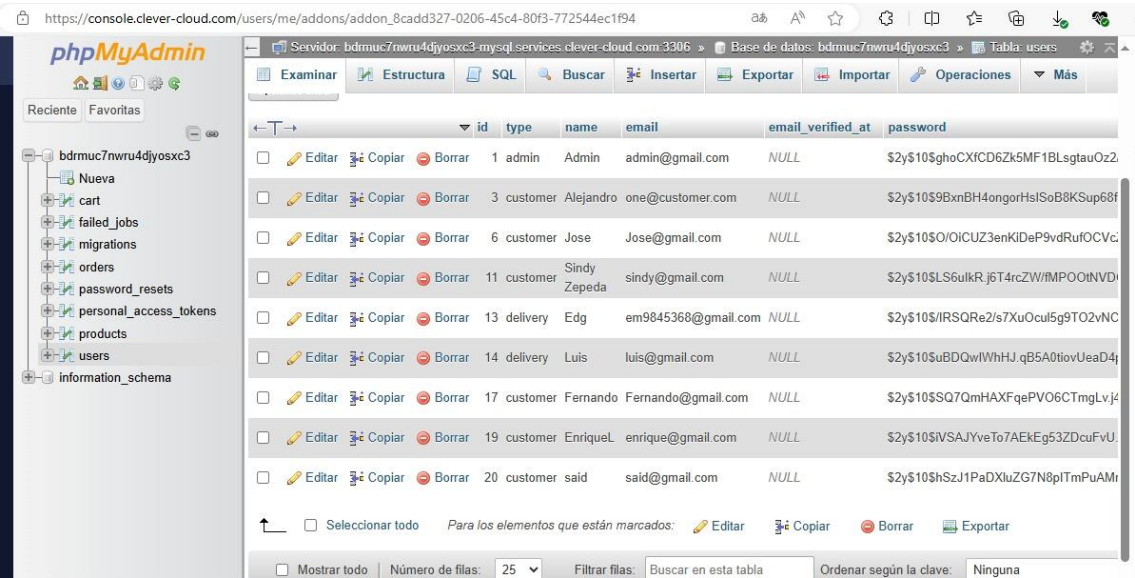
[Desplegar Laravel en el servidor Apache de Ubuntu - Código con Susan \(codewithsusan.com\)](#)



ANEXOS

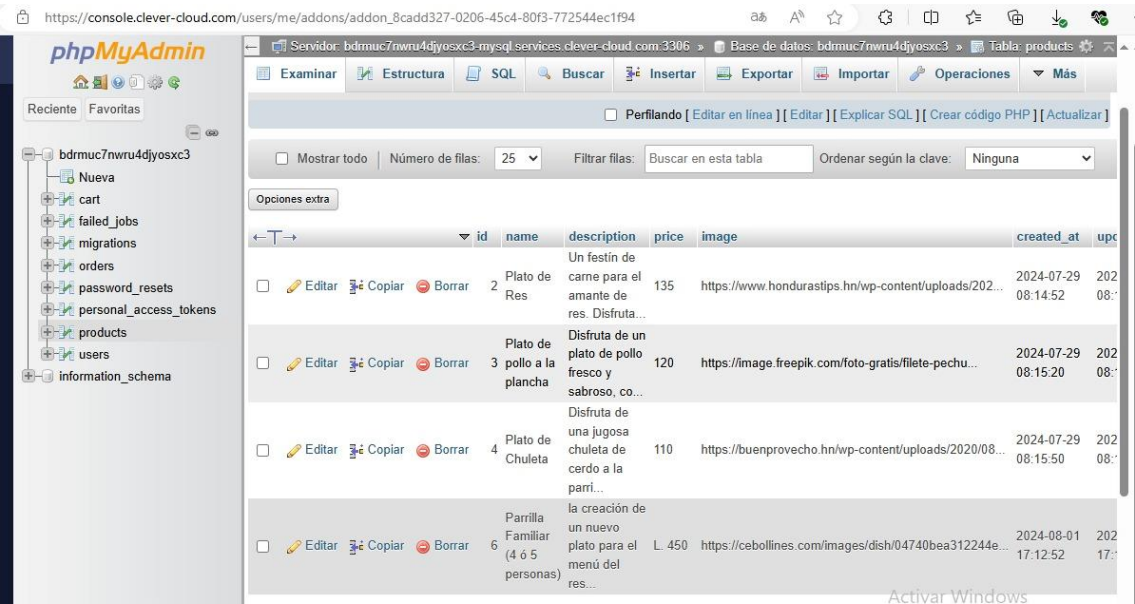
BASE DE DATOS

Tabla usuarios



	id	type	name	email	email_verified_at	password
<input type="checkbox"/>	1	admin	Admin	admin@gmail.com	NULL	\$2y\$10\$ghoCXICD6Zk5MF1BLsgtauOz2.
<input type="checkbox"/>	3	customer	Alejandro	one@customer.com	NULL	\$2y\$10\$9BxnBH4ongorHsISoB8KSup68f
<input type="checkbox"/>	6	customer	Jose	Jose@gmail.com	NULL	\$2y\$10\$O/OICUZ3enKiDeP9vdRuFOCVc.
<input type="checkbox"/>	11	customer	Sindy Zepeda	sindy@gmail.com	NULL	\$2y\$10\$LS6ulkRj6T4rcZWfMPOOtNVD.
<input type="checkbox"/>	13	delivery	Edg	em9845368@gmail.com	NULL	\$2y\$10\$/IRSQR2/s7XuOcul5g9TO2vNC.
<input type="checkbox"/>	14	delivery	Luis	luis@gmail.com	NULL	\$2y\$10\$uBDQwIWhHJ.qB5A0tiouvUeaD4.
<input type="checkbox"/>	17	customer	Fernando	Fernando@gmail.com	NULL	\$2y\$10\$SQ7QmHAXFqePVO6CTmgLvJ4.
<input type="checkbox"/>	19	customer	EnriqueL	enrique@gmail.com	NULL	\$2y\$10\$VSAJYveTo7AEKeg53ZDcuFvU.
<input type="checkbox"/>	20	customer	said	said@gmail.com	NULL	\$2y\$10\$hSzJ1PaDXluZG7N8pITmPuAMr.

Tabla producto



	id	name	description	price	image	created_at	updated_at
<input type="checkbox"/>	2	Plato de Res	Un festín de carne para el amante de res. Disfruta...	135	https://www.hondurastips.hn/wp-content/uploads/202...	2024-07-29 08:14:52	2024-07-29 08:14:52
<input type="checkbox"/>	3	Plato de pollo a la plancha	Disfruta de un plato de pollo fresco y sabroso, co...	120	https://image.freepik.com/foto-gratis/filete-pecu...	2024-07-29 08:15:20	2024-07-29 08:15:20
<input type="checkbox"/>	4	Plato de Chuleta	Disfruta de una jugosa chuleta de cerdo a la parri...	110	https://buenprovecho.hn/wp-content/uploads/2020/08...	2024-07-29 08:15:50	2024-07-29 08:15:50
<input type="checkbox"/>	6	Parrilla Familiar (4 ó 5 personas)	la creación de un nuevo plato para el menú del res...	L. 450	https://cebolines.com/images/dish/04740bea312244e...	2024-08-01 17:12:52	2024-08-01 17:12:52

Tabla ordenes

https://console.clever-cloud.com/users/me/addons/addon\_8cadd327-0206-45c4-80f3-772544ec1f94

phpMyAdmin

Reciente Favoritas

bdr muc7n wru4djiyosxc3

Nueva

cart

failed\_jobs

migrations

orders

password\_resets

personal\_access\_tokens

products

users

information\_schema

Examinar

Estructura

SQL

Buscar

Insertar

Exportar

Importar

Operaciones

Más

id

delivery\_boy\_email

customer\_email

items

destination\_address

destination\_lat

EditarCopiarBorrar

1

one@delivery.com

one@customer.com

[[{"item\_id": "1"}]]

San Vicente Centenario, 97325655

14.891192561670

EditarCopiarBorrar

2

Luis@gmail.com

one@customer.com

[[{"item\_id": "2"}]]

Macholola, 98051540

14.89368489497

EditarCopiarBorrar

3

em9845368@gmail.com

Jose@gmail.com

[[{"item\_id": "3"}, {"item\_id": "4"}]]

SVC, 8771819

14.89204251896

EditarCopiarBorrar

4

one@delivery.com

sindy@gmail.com

[[{"item\_id": "1"}]]

San Nicolás, 92737282

14.89070488422

EditarCopiarBorrar

5

em9845368@gmail.com

Luis@gmail.com

[[{"item\_id": "1"}, {"item\_id": "2"}, {"item\_id": "3"}]]

El salitre, 7273662

14.90228809885

EditarCopiarBorrar

6

em9845368@gmail.com

Jose@gmail.com

[[{"item\_id": "1"}, {"item\_id": "2"}]]

Sn,6177272

14.88764868006

EditarCopiarBorrar

7

em9845368@gmail.com

Fernando@gmail.com

[[{"item\_id": "1"}, {"item\_id": "2"}, {"item\_id": "3"}, {"item\_id": "4"}]]

San Vicente Centenario, 97325655

14.89148272065

EditarCopiarBorrar

8

luis@gmail.com

em9845368@gmail.com

[[{"item\_id": "1"}, {"item\_id": "2"}]]

Parque central, 81827

14.88852300347

EditarCopiarBorrar

9

em9845368@gmail.com

em9845368@gmail.com

[[{"item\_id": "4"}]]

ED

14.89045562974

EditarCopiarBorrar

10

em9845368@gmail.com

sindy@gmail.com

[[{"item\_id": "4"}]]

Nuevo cellac

14.89686203652

Ve a Configuración para activar Wind

42