

Documentación del Proyecto Laravel

Índice

- Documentación del Proyecto Laravel
 - Índice
- Resumen del Proyecto
- Estructura MVC
- Cambio de idioma en el proyecto
- Rutas Inicio Sesión (CLIENTE , DEALER , ADMINISTRADOR)
 - Modelos (Models)
 - Vistas (Views)
 - VISTAS EVENTOS
 - dashboard.blade.php
 - Vista con código JS
 - Código Blade + JavaScript
 - create.blade.php
 - Vista con código JS
 - edit.blade.php
 - modal_qr.blade.php
 - VISTAS INVITADOS
 - index.blade.php
 - Vista con código JS
 - create.blade.php
 - Vista con código JS
 - edit.blade.php
 - proteccion_datos.blade.php
 - update.blade.php
 - VISTAS ERRORES
 - Estructura de carpetas
 - Vistas comunes
 - VISTAS AUTH
 - Estructura de carpetas
 - Vistas comunes
 - Vistas destacadas
 - auth/login.blade.php
 - Vista con código JS
 - Objetivo
 - Código Blade + JavaScript
 - auth/register.blade.php
 - VISTAS ADMIN
 - Estructura de carpetas
 - Vistas comunes
 - VISTAS CLIENTE
 - Estructura de carpetas

- Vistas comunes
 - VISTAS LAYOUTS
- Estructura de carpetas
- Vistas comunes
 - VISTAS VENDOR
- Estructura de carpetas
- Vistas comunes
- Controlador (Controllers)
 - Auth/RegisteredUserController
 - Métodos principales
 - RUTAS RegisteredUserController
 - EventosController
 - Métodos principales
 - RUTAS EventosController
 - Datos extras funcion exportarInvitados():
 - InvitadosController
 - Métodos principales
 - RUTAS InvitadosController
 - CochesController
 - Métodos principales
 - RUTAS CochesController
 - TrazabilidadController
 - Métodos principales
 - RUTAS TrazabilidadController
 - AjustesController
 - Métodos principales
 - RUTAS AjustesController
 - ClienteController
 - Métodos principales
 - RUTAS ClienteController
 - PruebaDinamicaController
 - Métodos principales
 - RUTAS ReservaController
 - PruebaDinamicaController
 - Métodos principales
 - RUTAS PruebaDinamicaController
 - TimingController
 - Métodos principales
 - RUTAS TimingController
 - PatrocinadoresController
 - Métodos principales
 - RUTAS PatrocinadoresController
 - EventoConductorController
 - Métodos principales
 - RUTAS EventoConductorController
- Conclusión

- Logros principales:
- Recomendaciones para futuras mejoras:

Resumen del Proyecto

Este es un proyecto web desarrollado con **Laravel** para la gestión de eventos en una comunidad tipo **caravana**. La aplicación permite a los usuarios:

- **Registrarse e iniciar sesión** de forma segura utilizando **Laravel Sanctum**.
- Crear, ver, editar y eliminar **eventos** mediante una interfaz CRUD completa.
- Cada evento incluye información como nombre, descripción, fecha y ubicación.

El objetivo principal es facilitar la organización de actividades dentro de una comunidad móvil o viajera, brindando una plataforma simple y segura para gestionar eventos.

Estructura MVC

A continuación se describe cómo está estructurado el proyecto según el patrón **Modelo - Vista - Controlador (MVC)**.

Cambio de idioma en el proyecto

1. Publicar carpeta lang para ver las que tenemos por defecto (en) , se cambia por "es":

```
php artisan lang:publish
```

2. Ejecutar este comando:

```
composer require laravel-lang/common
```

3. Crear carpeta "es" con la traducción cambiada al español

```
php artisan lang:add es
```

4. Cambiar el .env para que se use la carpeta correcta , dejamos el "en" y usamos el "es":

```
APP_LOCALE=en , por:  
APP_LOCALE=es
```

Rutas Inicio Sesion (CLIENTE , DEALER , ADMINISTRADOR)

Carpeta	Vista	Ruta	Descripción
cliente/	login.blade.php	cliente/login	Ruta para cliente
dealer/	login.blade.php	dealer/login	Ruta para dealer
admin/	login.blade.php	admin.login	Ruta para admin

- Cada una son las rutas de acceso dependiendo del rol que tendra cada uno.

Modelos (Models)

- **User.php**
 - Representa a los usuarios registrados.
 - Campos principales:

```
protected $fillable = [  
    'rol',  
    'name',  
    'email',  
    'email_verified_at',  
    'password',  
    'remember_token'  
];
```

🔗 Relaciones en el modelo User

- **reservas()**
 - Tipo: **hasMany**
 - Un evento puede tener **muchas reservas**.
- **Evento.php**
 - Representa los eventos creados por los usuarios.
 - Campos principales:

```
protected $fillable = [
    'nombre',
    'marca',
    'fecha',
    'hora',
    'lugar_evento',
    'tipo_evento',
    'coste_evento',
    'aforo',
    'coste_unitario',
    'enlace',
    'documentacion',
    'texto_invitacion',
    'imagen'
];
```

🔗 Relaciones en el modelo Evento

```
public function invitados()
{
    return $this->belongsToMany(Conductor::class, 'evento_conductor', 'evento_id',
'conductor_id')
        ->withPivot([
            'cif',
            'nombre',
            'apellido',
            'email',
            'telefono',
            'empresa',
            'vehiculo_prop',
            'vehiculo_emp',
            'intolerancia',
            'preferencia',
            'carnet',
            'etiqueta',
            'kam',
            'asiste',
            'dni',
            'proteccion_datos',
            'carnet_caducidad',
            'confirmado',
            'token',
            'etiqueta_2'
        ]);
}

/**
 * La función `booted` en PHP asigna un `public_id` único a un evento si está
 * vacío, utilizando el evento `creating` de Laravel.
 */
```

```
protected static function booted()
{
    static::creating(function ($evento) {
        if (empty($evento->public_id)) {
            $evento->public_id = (string) \Illuminate\Support\Str::uuid();
        }
    });
}

public function marcas()
{
    return $this->belongsToMany(Marca::class, 'eventos_marca', 'evento_id',
'marca_id');
}

public function coches()
{
    return $this->hasMany(Coch::class, 'evento_id');
}

public function paradas()
{
    return $this->hasMany(Parada::class, 'evento_id')->orderBy('orden');
}

public function restaurante()
{
    return $this->hasOne(Restaurante::class, 'evento_id', 'id');
}

public function timings()
{
    return $this->hasMany(Timing::class, 'evento_id', 'id');
}

public function banners()
{
    return $this->hasMany(Banner::class, 'evento_id', 'id');
}
/**
 * La función `getRouteKeyName()` en PHP devuelve el nombre de la clave de ruta
 'public_id'.
 *
 * @return El método `getRouteKeyName()` devuelve la cadena `public_id`. Este
 método se utiliza en
 * Laravel para especificar el atributo que se debe usar al recuperar un modelo por
 su clave de ruta. En
 * este caso, la clave de ruta para el modelo será el atributo `public_id`.
 */
public function getRouteKeyName()
{
    return 'public_id';
}
```

- **Conductor.php**
- Representa los invitados creados por los usuarios.
 - Campos principales:

```
protected $fillable = [
    'cif',
    'nombre',
    'apellido',
    'email',
    'telefono',
    'empresa',
    'vehiculo_prop',
    'vehiculo_emp',
    'intolerancia',
    'preferencia',
    'carnet',
    'etiqueta',
    'kam',
    'asiste',
    'dni',
    'proteccion_datos',
    'carnet_caducidad'
];
```

🔗 Relaciones en el modelo Conductor

- **eventos()**
 - Tipo: `belongsToMany`
 - Un conductor puede estar **en varios** eventos.
 - Relación con el modelo `Evento`.
 - Usa la tabla pivot `evento_conductor` con las claves: `conductor_id` y `evento_id`.

```
public function eventos()
{
    return $this->belongsToMany(Evento::class, 'evento_conductor',
'conductor_id', 'evento_id')
        ->withPivot([
            'cif',
            'nombre',
            'apellido',
            'email',
            'telefono',
            'empresa',
            'vehiculo_prop',
            'vehiculo_emp',
            'intolerancia',
            'preferencia',
```

```

        'carnet',
        'etiqueta',
        'kam',
        'asiste',
        'dni',
        'proteccion_datos',
        'carnet_caducidad',
        'confirmado',
        'token',
        'etiqueta_2'
    ]);
}

public function conductores()
{
    return $this->belongsToMany(Conductor::class, 'evento_conductor', 'evento_id',
'conductor_id');
}

```

- **EventoConductor.php**
- Representa tabla intermedia entre modelos Evento y Conductor.
 - Campos principales:

```

protected $fillable = [
    'evento_id',
    'conductor_id',
    'confirmado',
    'token'
];

```

🔗 Relaciones en el modelo EventoConductor

- **invitados()**
 - Tipo: **hasMany**
 - Un evento puede tener varios conductores asociados.
 - Relación con el modelo **Conductor**.

```

public function invitados(){
    return $this->hasMany(Conductor::class);
}

```

- **EventosMarca.php**
- Representa tabla intermedia entre modelos Evento y Marcas.
 - Campos principales:

```
protected $fillable = [
    'evento_id',
    'marca_id'
];
```

- **Marca.php**
- Representa los nombres de las marcas que tengo guardados en la BD.
 - Campos principales:

```
protected $fillable = [
    'nombre'
];
```

🔗 Relaciones en el modelo Evento

- **eventos()**
 - Tipo: `belongsToMany`
 - Una marca puede estar asociada cuando se crea el evento.
 - Relación con el modelo `Evento`.
 - Usa la tabla pivot `eventos_marca` con las claves: `marca_id` y `evento_id`.

```
public function eventos(){
    return $this->belongsToMany(Evento::class , 'eventos_marca' , 'marca_id' ,
'evento_id');
```

- **TipoEvento.php**
- Representa tabla del tipo de evento que tengo en mi BD
 - Campos principales:
- **Coch.php**
- Representa la información de los coches que tengo guardados en la BD.

- Campos principales:

```
protected $fillable = [
    'marca',
    'modelo',
    'version',
    'matricula',
    'kam',
    'asiste',
    'evento_id',
    'seguro',
    'documento_seguro',
    'foto_vehiculo'
];
```

🔗 Relaciones en el modelo Coch

```
public function reservas()
{
    return $this->hasMany(Reserva::class, 'coche_id');
}

public function evento()
{
    return $this->belongsTo(Evento::class, 'evento_id');
}
```

- **Parada.php**
- Representa la informacion de las paradas que tengo guardados en la BD de cada evento.
 - Campos principales:

```
protected $fillable = [
    'evento_id',
    'conductor',
    'nombre',
    'descripcion',
    'enlace',
    'orden'
];
```

🔗 Relaciones en el modelo Parada

```

public function evento()
{
    return $this->belongsTo(Evento::class, 'evento_id');
}

public function reservas()
{
    return $this->hasMany(Reserva::class);
}

```

- **Reserva.php**

- Representa la informacion de las reservas que tengo guardadas en la BD de cada evento , como usuario.
 - Campos principales:

```

protected $fillable = [
    'user_id',
    'coche_id',
    'parada_id',
    'evento_id',
    'tipo'
];

protected $casts = [
    'user_id' => 'int',
    'coche_id' => 'int',
    'parada_id' => 'int',
    'evento_id' => 'int',
    'hora_inicio' => 'string',
    'hora_fin' => 'string'
];

```

🔗 Relaciones en el modelo Reserva

```

public function coche()
{
    return $this->belongsTo(Coch::class, 'coche_id');
}

public function parada()
{
    return $this->belongsTo(Parada::class , 'parada_id');
}

public function user()
{
    return $this->belongsTo(User::class , 'user_id');
}

```

```
}

    public function evento(){
        return $this->belongsTo(Evento::class , 'evento_id');
    }
}
```

- **Restaurante.php**
- Representa la informacion de los o el restaurante que tengo guardado en la BD de cada evento.
 - Campos principales:

```
protected $casts = [
    'evento_id' => 'int'
];

protected $fillable = [
    'nombre',
    'descripcion',
    'foto_restaurante',
    'enlace',
    'evento_id'
];
```

🔗 Relaciones en el modelo Restaurante

```
public function evento()
{
    return $this->belongsTo(Evento::class, 'evento_id' , 'id');
}
```

- **Timing.php**
- Representa la informacion del timing que tengo guardado en la BD de cada evento.
 - Campos principales:

```
protected $fillable = [
    'nombre',
    'descripcion',
    'evento_id'
];
```

🔗 Relaciones en el modelo Timing

```

public function timings()
{
    return $this->hasMany(Timing::class);
}

public function evento()
{
    return $this->belongsTo(Evento::class , 'evento_id', 'id');
}

```

- **Banner.php**

- Representa la informacion de cada banner(empresa) que tengo guardado en la BD de cada evento.
 - Campos principales:

```

protected $fillable = [
    'evento_id',
    'empresa',
    'enlace',
    'video',
    'imagen',
    'frase',
    'contacto',
    'texto'
];

```

🔗 Relaciones en el modelo Banner

```

public function evento(){
    return $this->belongsTo(Evento::class, 'evento_id');
}

```

⌚ Vistas (Views)

Esta sección describe la estructura y organización de las vistas dentro del proyecto.

VISTAS EVENTOS

dashboard.blade.php

- **Propósito:** Pagina de inicio al entrar.
- **Datos requeridos:** For each con distintos campos creados para mostrar todo el contenido (se pasan en el controlador).
- **Acciones:** La vista dashboard esta comentada para que puedas leer el contenido de cada una de sus partes , a continuacion te explico lo mas importante.

- **Extras:** La vista cuenta con comentarios donde se especifica concretamente cada parte del código , de tal manera que se vea mas legible.

✍ Vista con código JS

A continuación, se muestra un ejemplo de cómo se puede incluir una función en JavaScript dentro de una vista Blade para limpiar y refrescar la página

❖ Código Blade + JavaScript

```
<!-- CONTENEDOR DE BOTONES (FORMULARIO) -->
<form class="d-block flex-wrap col-xl-10" action="{{ route('eventos.show') }}" method="GET">
    <div class="d-flex flex-wrap col-lg-12 align-items-center">
        <div class="col-12 col-lg-6">
            //Campo de búsqueda ( se usa para el buscador).
            <input class="form-control mb-3" id="nombre" name="buscador" type="text" placeholder="Introducir nombre y otros campos">
        </div>
        <div class="col-12 col-lg-6 d-flex gap-2 justify-content-between">
            <button type="submit" class="btn_color txt col-4 col-sm-4 col-lg-4 mb-3 mx-lg-2">Buscar</button>
            <button type="submit" class="btn_color txt col-3 btn_secundario col-sm-4 col-lg-4 mb-3" id="reset">Limpiar</button>
            <a href="{{ route('eventos.create') }}" class="btn_color col-4 col-sm-4 col-lg-4 mb-3">Crear evento</a>
        </div>
    </div>
</form>

<!-- LIMPIAR CAMPO-->
<script>
// Campos de búsqueda (DOM)
const input_nombre = document.getElementById('nombre');
const btn_reset = document.getElementById('reset');

// Función limpiar.
function reset() {
    input_nombre.value = "";
    window.location = "/";
}

btn_reset.addEventListener('click', (e) => {
    e.preventDefault();
    reset();
})
</script>
```

- **Propósito:** Pagina para crear evento.
- **Datos requeridos:** For each con distintos campos creados para mostrar todo el contenido (se pasan en el controlador).
- **Extras:** La vista cuenta con comentarios donde se especifica concretamente cada parte del código , de tal manera que se vea mas legible.

Vista con código JS

A continuación, se muestra un ejemplo de cómo se puede incluir una función en JavaScript dentro de una vista Blade para calcular la media y obtener resultado del COSTE UNITARIO..

```
//CAMPOS DE CADA INPUT.
<!-- COSTE EVENTO -->
<input type="text" id="coste_evento" class="form-control" name="coste_evento"
placeholder="Coste evento" oninput="calcular_media()">

<!-- AFORO MÁXIMO -->
<input type="text" id="aforo_maximo" class="form-control" name="aforo"
placeholder="Aforo máximo" oninput="calcular_media()">

<!-- COSTE UNITARIO -->
<input type="text" id="coste_unitario" class="form-control" name="coste_unitario"
placeholder="Coste unitario">

<!--COSTE UNITARIO (FUNCION JS)-->
<script>
    const coste_evento = document.getElementById('coste_evento');
    const aforo_maximo = document.getElementById('aforo_maximo');
    const coste_unitario = document.getElementById('coste_unitario');

    function calcular_media() {
        let coste = parseFloat(coste_evento.value);
        let aforo = parseFloat(aforo_maximo.value);
        let media = coste / aforo;

        if (!isNaN(media) && isFinite(media)) {
            coste_unitario.value = media.toFixed(2);
        } else {
            coste_unitario.value = '';
        }
    }
</script>
```

edit.blade.php

- **Propósito:** Pagina para editar evento.
- **Datos requeridos:** For each con distintos campos creados para mostrar todo el contenido (se pasan en el controlador).

- **Extras:** La vista cuenta con comentarios donde se especifica concretamente cada parte del código, de tal manera que se vea más legible. Suele ser igual la vista de create.blade.php pero en este caso recogemos valores que se mostrarán en el formulario (GET).

```
//En el formulario se pasa el "enctype" debido a que subiremos archivos e imágenes
//se explican y se pasan luego en el controlador).
<form method="POST" action="{{route('eventos.update', $evento->id)}}" class="m-
auto mt-5 mb-5" style="width: 70%;" enctype="multipart/form-data">

//Tener en cuenta esta sección ya que depende del modelo Eloquent de MARCA y
EVENTO.
<!--MARCA-->
<div class="mb-3">
    <label for="marca">Marca* (Para seleccionar más de una opción, pulsa la
tecla Control)</label>
    //Pasamos marca como array ya que en la tabla intermedia se recogen los
valores.
    <select class="form-select" name="marca[]" multiple>
        @if(isset($marcas))
            //sortBy ordena los nombres en orden alfabético.
            @foreach($marcas->sortBy('nombre') as $marca)
                //Recoge el valor del id de la tabla marcas.
                <option value="{{{$marca->id}}}">
                    //Luego lo busca y lo selecciona en el option.
                    {{ $evento->marcas->pluck('id')->contains($marca->id) ? 'selected'
: '' }}>
                    {{$marca->nombre}}
                </option>
            @endforeach
            @endif
        </select>
    </div>
```

modal_qr.blade.php

- **Propósito:** Página donde se encuentra el modal del código QR..
- **Acciones:** La vista modal_qr está comentada para que puedas leer el contenido de cada una de sus partes.
- **Extras:** La vista cuenta con comentarios donde se especifica concretamente cada parte del código, de tal manera que se vea más legible.

VISTAS INVITADOS

index.blade.php

- **Propósito:** Página principal de invitados.

- **Datos requeridos:** For each con distintos campos creados para mostrar todo el contenido (se pasan en el controlador).
- **Extras:** La vista cuenta con comentarios donde se especifica concretamente cada parte del código , de tal manera que se vea mas legible , esta vista cuenta con dos maneras de ver el contenido de los invitados. , una CARD para tamaño SM Y MD y otra que es la tabla que se mostrara para tamaño LG Y XL (responsive).

📝 Vista con código JS

A continuación, se muestra un ejemplo de cómo se puede seleccionar en un CHECKBOX si el invitado ASISTE o NO ASISTE , tambien te explico el JS de como poder eliminar un invitado de ese mismo evento.

```
<!--BOTON ELIMINAR-->
<form action="{{route('invitados.delete', $invitado->id)}}" method="POST">
    @csrf
    @method('DELETE')
    //El boton recoge informacion del id(invitado) mediante un nombre que le
    ponemos.
    <button name="delete" id_delete="{{$invitado->id}}" type="submit"
    style="background:none; border:none; color:#05072e;">
        <i class="bi bi-trash3-fill"></i>
    </button>
</form>

//FUNCION PARA ELIMINAR
document.addEventListener('DOMContentLoaded', function() {
    //Pasamos al DOM el boton para cada invitado mediante su nombre , con
    querySelectorAll seleccionamos todos.
    const btn_delete = document.querySelectorAll('[name="delete"]');

    btn_delete.forEach((element) => {
        element.addEventListener('click', (e) => {
            e.preventDefault();
            Swal.fire({
                title: "¿Seguro que quieres eliminar?",
                icon: "warning",
                iconColor: "#05072e",
                showDenyButton: true,
                denyButtonColor: "#05072e",
                showCancelButton: false,
                showCancelColor: "#05072e",
                confirmButtonText: "Sí",
                confirmButtonColor: "#05072e",
                denyButtonText: `No`
            }).then((result) => {
                if (result.isConfirmed) {
                    //Recogemos el campo id_delete mediante funcion
                   getAttribute.
                    let id = element.getAttribute('id_delete');
                    //Pasamos en el fetch la ruta y se reemplaza el:id por id
                    para que me reciba la peticion.
                }
            })
        })
    })
})
```

```
        fetch(`{{route('invitados.delete',
':id')}}` .replace(':id', id), {
            method: 'DELETE',
            //Token de seguridad.
            headers: {
                'X-CSRF-TOKEN':
document.querySelector('meta[name="csrf-token"]').getAttribute('content')
            }
        })
        .then(response => response.json())
        .then(data => {
            //Si es exitoso.
            if (data.success) {
                Swal.fire({
                    title: "¡Eliminado!",
                    icon: "success",
                    iconColor: "#05072e",
                    showConfirmButton: true,
                    confirmButtonColor: "#05072e"
                })
                .then(() => {
                    window.location.href =
{{route('invitados.index' , $evento->id)}}";
                })
                //Si falla.
            } else {
                Swal.fire("Error al eliminar.", "", "error");
            }
        })
        //Error
        .catch(error => {
            console.log("Respuesta");
            Swal.fire("Error al eliminar.", "", "error");
        });
        //Si damos al no los cambios no se guardan.
    } else if (result.isDenied) {
        Swal.fire({
            title: "Cambios no guardados.",
            icon: "warning",
            iconColor: "#05072e",
            confirmButtonColor: "#05072e"
        });
    }
});
});;
});;
});;
});;

//FUNCION CHECKBOX ASISTENCIA
document.addEventListener('DOMContentLoaded', () => {
    const token = document.querySelector('meta[name="csrf-
token"]')).getAttribute('content');
```

```
document.querySelectorAll('.js-asiste').forEach(chk => {
    chk.addEventListener('change', async (e) => {
        const el = e.currentTarget;
        const eventoId = el.dataset.eventoId;
        const invitadoId = el.dataset.invitadoId;
        const valor = el.checked;

        try {
            const res = await
fetch(`/eventos/${eventoId}/invitados/${invitadoId}/asistencia`, {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json',
                    'X-CSRF-TOKEN': token
                },
                body: JSON.stringify({
                    asiste: valor
                })
            });
            if (!res.ok) throw new Error('HTTP ' + res.status);

            const data = await res.json();
            if (!data.success) throw new Error('Respuesta inválida');

            // refresca contadores
            const t = document.getElementById('total-span');
            const a = document.getElementById('asisten-span');
            const na = document.getElementById('noasiste-span');
            if (data.totals) {
                if (t) t.textContent = data.totals.total;
                if (a) a.textContent = data.totals.asisten;
                if (na) na.textContent = data.totals.no_asiste;
            }
        } catch (err) {
            el.checked = !valor; // revierte
            if (window.Swal) {
                Swal.fire({
                    title: "Error al actualizar asistencia.",
                    text: err.message || "",
                    icon: "warning",
                    confirmButtonColor: "#05072e"
                });
            } else {
                alert('Error al actualizar asistencia');
            }
        }
    });
});
});
```

create.blade.php

- **Propósito:** Pagina para crear invitados.
- **Datos requeridos:** For each con distintos campos creados para mostrar todo el contenido (se pasan en el controlador).
- **Extras:** La vista cuenta con comentarios donde se especifica concretamente cada parte del código , de tal manera que se vea mas legible.

```
<!--VEHICULO PROPIO-->
<div class="mb-5 col-md-8">
    <label for="vehiculo_prop" class="form-label fw-bold" style="margin-right: 2%">¿Cuenta con vehiculo de uso propio?</label>
    <div class="form-check">
        <input id="btn_si_prop" class="form-check-input" type="radio" value="si" name="vehiculo_prop" {{ old('vehiculo_prop') == 'si' ? 'checked' : '' }}>
        <label class="form-check-label" for="vehiculo_prop">Sí</label>
    </div>
    <div class="form-check">
        <input id="btn_no_prop" class="form-check-input" type="radio" value="no" name="vehiculo_prop" {{ old('vehiculo_prop') == 'no' ? 'checked' : '' }}>
        <label class="form-check-label" for="vehiculo_prop">No</label>
    </div>
</div>

<!--ETIQUETA 1 (MOSTRAR)-->
<div id="etiqueta-container_prop" style="display: none;" class="mx-md-2">
<!--ETIQUETA 2 (MOSTRAR)-->
<div id="etiqueta-container_emp" style="display: none;" class="mx-md-2">

<!--VEHICULO EMPRESA-->
<div class="mb-5 col-md-8">
    <label for="vehiculo_emp" class="form-label fw-bold" style="margin-right: 2%;">¿Cuenta con un vehiculo de empresa?</label>
    <div class="form-check">
        <input id="btn_si_emp" class="form-check-input" type="radio" value="si" name="vehiculo_emp" {{ old('vehiculo_emp') == 'si' ? 'checked' : '' }}>
        <label class="form-check-label" for="btn_si_emp">Sí</label>
    </div>
    <div class="form-check">
        <input id="btn_no_emp" class="form-check-input" type="radio" value="no" name="vehiculo_emp" {{ old('vehiculo_emp') == 'no' ? 'checked' : '' }}>
        <label class="form-check-label" for="btn_no_emp">No</label>
    </div>
</div>
```

✍ Vista con código JS

A continuación, se muestra un ejemplo de cómo se puede seleccionar en un CHECKBOX si el invitado cuenta con coche de vehículo empresa o coche propio , actualmente esta parte esta hecha para que solo pueda ser si por una de sus partes ya que un invitado solo podrá tener un solo coche.

```
<!--JS ETIQUETA-->
<script>
    const boton_si_prop = document.getElementById('btn_si_prop');
    const boton_no_prop = document.getElementById('btn_no_prop');
    const etiquetaContainerProp = document.getElementById('etiqueta-
container_prop');

    function mostrar() {
        if (boton_si_prop.checked) {
            etiquetaContainerProp.style.display = "block";
        } else {
            etiquetaContainerProp.style.display = "none";
        }
    }

    boton_si_prop.addEventListener('click', () => {
        boton_no_emp.checked = false;
        mostrar();
    })

    boton_no_prop.addEventListener('click', () => {
        mostrar();
    });
</script>
```

edit.blade.php

- **Propósito:** Pagina para editar invitados.
- **Datos requeridos:** For each con distintos campos creados para mostrar todo el contenido (se pasan en el controlador).
- **Extras:** La vista cuenta con comentarios donde se especifica concretamente cada parte del código , de tal manera que se vea mas legible. Suele ser igual la vista de create.blade.php pero en este caso recogemos valores que se mostraran en el formulario (GET).

proteccion_datos.blade.php

- **Propósito:** Código en HTML que usare para aplicar al modal en el Protección de Datos en el formualrio.

update.blade.php

- **Propósito:** Este formulario servira para poder pooder mandar el correo de registro al invitado.

VISTAS ERRORES

 Estructura de carpetas

 Vistas comunes

Carpeta	Vista	Descripción
---------	-------	-------------

Carpeta	Vista	Descripción
errors/401	401.blade.php	Acceso no autorizado
errors/404	404.blade.php	URL no existe.

- **Propósito:** Paginas principales que mostraran errores que puedan surgir.
- **Extras:** La vista cuenta con comentarios donde se especifica concretamente cada parte del codigo.

VISTAS AUTH

📁 Estructura de carpetas

❖ Vistas comunes

Carpeta	Vista	Descripción
auth/	confirm-password.blade.php	Confirma contraseña
auth	forgot.blade.php	Olvida contraseña
auth/	login.blade.php	Inicio sesion
auth/	register.blade.php	Registro
auth	reset-password.blade.php	Cambiar contraseña
auth/	verify-email.blade.php	Verificar email

- **Propósito:** Paginas principales de autenticacion que serviran para acceder a la aplicacion como administrador entre otras.
- **Extras:** La vista cuenta con comentarios donde se especifica concretamente cada parte del codigo.

📄 Vistas destacadas

auth/login.blade.php

- **Propósito:** Poder loguearte al entrar en la pagina del dashboard.
- **Datos requeridos:** Marcar la casilla de politica de privacidad y normas de uso.
- **Acciones:** Permitir validacion antes de entrar.

📝 Vista con código JS

A continuación, se muestra un ejemplo de cómo se puede incluir una validación en JavaScript dentro de una vista Blade para asegurarse de que el usuario acepte los términos antes de iniciar sesión.

☑ Objetivo

Evitar que el formulario se envíe si el checkbox de "He leído y acepto las normas de uso" no está marcado y funcion tamnbien para mostrar y ocultar.

❖ Código Blade + JavaScript

```
<!-- Remember Me -->
<div class="mb-3">
    <label for="remember_me" class="inline-flex items-center">
        <input id="remember_me" type="checkbox" class="rounded border-gray-300 text-indigo-600 shadow-sm focus:ring-indigo-500" name="remember">
        <span class="ms-2 text-sm text-gray-600">{{ __('He leído y acepto las normas de uso y la política de privacidad.') }}</span>
    </label>
</div>

<!-- VALIDACIÓN JS PARA PODER LOGUEARTE -->
<script>
    const remember = document.getElementById('remember_me');
    const form = document.getElementById('form');

    function validarCheckbox() {
        if (!remember.checked) {
            Swal.fire({
                title: "Marque casilla para continuar.",
                icon: "warning",
                iconColor: "#05072e",
                confirmButtonText: "Aceptar",
                confirmButtonColor: "#05072e"
            });
            return false;
        }
        return true;
    }

    form.addEventListener('submit', (e) => {
        if (!validarCheckbox()) {
            e.preventDefault();
        }
    });
</script>

<!----FUNCION PARA MOSTRAR/OCULTAR CONTRASEÑA(JS)-->

<div class="input-group">
    <span id="toggle-password" class="input-group-text"><i class="bi bi-eye-fill"></i></span>
    <x-text-input id="password" type="password" name="password" required
    autocomplete="current-password" class="form-control" />
</div>

<script>
    const inputPassword = document.getElementById('password'); //Campo texto de contraseña.
    const togglePassword = document.getElementById('toggle-password'); //Icono imagen contraseña para mostrar y ocultar.
    const icon = togglePassword.querySelector('i');//Icono
```

```

function mostrar() {
    //Si el campo de texto es tipo password que me cambie el icono y me
    muestre la contraseña(mostrar).
    if (inputPassword.type === 'password') {
        inputPassword.type = 'text';
        icon.classList.remove('bi-eye-fill');
        icon.classList.add('bi-eye-slash-fill');

    } else {
        //Sino que me muestre todo lo contrario y lo deje como tipo
        password(ocultar)
        inputPassword.type = 'password';
        icon.classList.remove('bi-eye-slash-fill');
        icon.classList.add('bi-eye-fill');
    }
}

//Cuando le de click que me realize la funcion.
togglePassword.addEventListener('click', () => {
    mostrar();
})
</script>

```

auth/register.blade.php

- **Propósito:** Poder registrarte en la pagina.

```

@extends('layouts.login')
@section('content')
<x-guest-layout>
    <form method="POST" id="formulario" action="{{ route('register') }}"
    class="col-12">
        @csrf

        <!-- Name (opcional) -->
        <div>
            <x-input-label for="name" :value="__(‘Nombre’)"/>
            <x-text-input id="name" class="block mt-1 w-full validar" type="text"
            name="name" :value="old(‘name’)" autofocus autocomplete="name" />
            <x-input-error :messages="$errors->get(‘name’) " class="mt-2" />
        </div>

        <!-- Email Address -->
        <div class="mt-4">
            <x-input-label for="email" :value="__(‘Email’)"/>
            <x-text-input
                id="email"
                class="block mt-1 w-full validar"

```

```
        type="email"
        name="email"
        value="{{ old('email') }}"
        autocomplete="username" />
    </div>

    <!-- Password -->
    <div class="mt-4">
        <x-input-label for="password" :value="__( 'Contraseña' )" class="fw-
bold" />

        <div class="input-group">
            <x-text-input type="password" name="password" autocomplete="new-
password" class="form-control password validar" />
            <span class="input-group-text toggle-password" style="cursor:
pointer;">
                <i class="bi bi-eye-fill"></i>
            </span>
        </div>
        <x-input-error :messages="$errors->get('password')" class="mt-2" />
    </div>

    <!-- Confirm Password -->
    <div class="mt-4">
        <x-input-label for="password_confirmation" :value="__( 'Confirmar
contraseña' )" />
        <div class="input-group">
            <x-text-input type="password" name="password_confirmation"
autocomplete="new-password" class="form-control password validar" />
            <span class="input-group-text toggle-password" style="cursor:
pointer;">
                <i class="bi bi-eye-fill"></i>
            </span>
        </div>
        <x-input-error :messages="$errors->get('password_confirmation')"
class="mt-2" />
    </div>

    <!--SWEETALERT PARA EL CORREO-->
@if ($errors->has('email'))
<script>
    document.addEventListener('DOMContentLoaded', function() {
        Swal.fire({
            icon: 'error',
            iconColor: "#05072e",
            title: 'Correo no existe en la BD',
            confirmButtonColor: "#05072e"
        });
    });
</script>
@endif

<script>
    document.addEventListener('DOMContentLoaded', function() {
```

```
// Toggle de contraseña
const toggles = document.querySelectorAll('.toggle-password');
toggles.forEach(toggle => {
    toggle.addEventListener('click', function() {
        const input = this.previousElementSibling;
        const icon = this.querySelector('i');

        if (input.type === 'password') {
            input.type = 'text';
            icon.classList.remove('bi-eye-fill');
            icon.classList.add('bi-eye-slash-fill');
        } else {
            input.type = 'password';
            icon.classList.remove('bi-eye-slash-fill');
            icon.classList.add('bi-eye-fill');
        }
    });
});

// Validación del formulario
const form = document.getElementById('formulario');
const inputs = document.querySelectorAll('.validar');

form.addEventListener('submit', function(e) {
    // validación front (rápida) para UX; el back manda de verdad
    let valido = true;

    inputs.forEach(function(input) {
        if (!input.value || input.value.trim() === "") {
            input.classList.add('validacion-mal');
            input.classList.remove('validacion-bien');
            valido = false;
        } else {
            input.classList.remove('validacion-mal');
            input.classList.add('validacion-bien');
        }
    });

    if (!valido) {
        e.preventDefault();
    }
});

// Eliminar clase de error en tiempo real
inputs.forEach(function(input) {
    input.addEventListener('input', function() {
        if (input.value.trim() !== "") {
            input.classList.remove('validacion-mal');
        }
    });
});

```

```

<div class="flex items-center justify-end mt-4 gap-2">
    <a class="underline text-sm text-gray-600 hover:text-gray-900 rounded-md focus:outline-none focus:ring-2 focus:ring-offset-2 focus:ring-indigo-500"
        href="{{ route('login') }}"
        {{ __('Ya registrado?') }}
    </a>

    <button type="submit" class="ms-4 px-4 py-2 bg-indigo-600 text-white rounded-md hover:bg-indigo-700 btn_color">
        {{ __('Registrarse') }}
    </button>
</div>
</form>
</x-guest-layout>
@endsection

```

VISTAS ADMIN

Estructura de carpetas

Vistas comunes

Carpeta	Vista	Descripción
admin/modals	modal_banner.blade.php	Modal banner
admin/modals	modal_coche.blade.php	Modal coches
admin/modals	modal_parada.blade.php	Modal parada
admin/modals	modal_restaurante.blade.php	Modal restaurante
admin/modals	modal_timing.blade.php	Modal timing
admin/	ajuste.blade.php	Vista dashboard Ajustes (se incluyen los modales)

- **Extras:** La vista cuenta con comentarios donde se especifica concretamente cada parte del código.

VISTAS CLIENTE

Estructura de carpetas

Vistas comunes

Carpeta	Vista	Descripción
cliente/modals	modal_info_restaurante.blade.php	Modal para mostrar información restaurante
cliente/modals	modal_info_ruta.blade.php	Modal para mostrar información ruta
cliente/modals	modal_info_patrocinadores.blade.php	Modal para mostrar partners (patrocinadores)

Carpeta	Vista	Descripción
cliente/modals	modal_pruebaDinamica.blade.php	Modal para realizar la prueba dinamica
cliente/modals	modal_reserva.blade.php	Modal para realizar la pre-reserva
cliente/	modal_timing.blade.php	Modal para mostrar el timing (horario)
cliente/	dashboard.blade.php	Pagina principal para la vista cliente
cliente/	info_coches.blade.php	Muestra la informacion de los coches del evento
cliente/	login.blade.php	Formulario de login para CLIENTE
cliente/	ruta.blade.php	Muestra la informacion de la ruta del evento

- **Extras:** La vista cuenta con comentarios donde se especifica concretamente cada parte del codigo.

VISTAS LAYOUTS

📁 Estructura de carpetas

❖ Vistas comunes

Carpeta	Vista	Descripción
layouts	admin.blade.php	Esqueleto vista admin , incluye CSS
layouts	cliente.blade.php	Esqueleto vista cliente , incluye CSS
layouts	coches.blade.php	Esqueleto vista coches , incluye CSS
layouts	error.blade.php	Esqueleto vista coches , incluye CSS
layouts	footer.php	Esqueleto footer , incluye CSS
layouts/	login.php	Esqueleto login , incluye CSS
layouts/	main.blade.php	Esqueleto principal , incluye CSS
layouts/	navigation.blade.php	Esqueleto navegacion , incluye CSS

- **Extras:** La vista cuenta con comentarios donde se especifica concretamente cada parte del codigo.

VISTAS VENDOR

📁 Estructura de carpetas

❖ Vistas comunes

Carpeta	Vista	Descripción
vendor	mail/html	Esqueleto correos CSS

Carpeta	Vista	Descripción
vendor	mail/text	Esqueleto correos CSS
vendor	notifications/email.blade.php	Esqueleto correos CSS

- **Extras:** La vista cuenta con comentarios donde se especifica concretamente cada parte del código.

② Controlador (Controllers)

❖ Descripción general

Los controladores son responsables de recibir las solicitudes del usuario, procesar la lógica correspondiente (generalmente delegando en modelos o servicios), y retornar una respuesta (normalmente una vista o JSON).

En este proyecto, los controladores siguen el patrón MVC proporcionado por Laravel y se encuentran en la ruta:

```
app/Http/Controllers/
```

📁 Estructura de controladores

EventosController

Encargado de manejar toda la lógica relacionada con la gestión de eventos: mostrar, crear, editar y eliminar eventos.

InvitadosController

Encargado de manejar toda la lógica relacionada con la gestión de invitados: mostrar, crear, editar y eliminar invitados.

AuthController

Controla los procesos de autenticación: login, logout, registro, etc.

Auth/RegisteredUserController

❖ Métodos principales

Método	Descripción
create()	Muestra la ventana de creación para poder visualizar el formulario.
store(Request \$request)	Permite la creación de un nuevo usuario.

RUTAS RegisteredUserController

```
Route::get('register', [RegisteredUserController::class, 'create'])
    ->name('register');
```

```
Route::post('register', [RegisteredUserController::class, 'store']);
```

EventosController

❖ Métodos principales

Método	Descripción
index()	Muestra una vista con la lista de eventos paginados, tipos de evento y marcas.
create()	Muestra el formulario.
store()	Guarda un nuevo evento en la base de datos.
edit(\$id)	Muestra el formulario para editar un evento existente.
update(Request \$request, \$id)	Actualiza la información de un evento.
delete(\$id)	Elimina un evento.
show(Request \$request)	Busca por todos los campos en el buscador.
filtrarFecha(Request \$request)	Filtre por fecha de inicio y fecha de fin (rango).
exportarInvitado(\$evento_id)	Exporta en documento Excel los invitados de cada evento.
conductorForm(Evento \$evento)	Muestra formulario que usaremos para el registro de sin LOGIN.
conductorStore(Request \$request , Evento \$evento)	Funcion que se usara para el registro de los conductores sin LOGIN.

RUTAS EventosController

```
/**CRUD EVENTOS */
Route::get('/eventos/create', [EventosController::class, 'create'])-
>name('eventos.create');
Route::post('/eventos/store', [EventosController::class, 'store'])-
>name('eventos.store');
Route::delete('/eventos/delete/{id}', [EventosController::class, 'delete'])-
>name('eventos.delete');
Route::get('/eventos/edit/{id}', [EventosController::class, 'edit'])-
>name('eventos.edit');
Route::put('/eventos/update/{id}', [EventosController::class, 'update'])-
>name('eventos.update');

Route::get('/eventos/show', [EventosController::class, 'show'])-
>name('eventos.show');
Route::get('/eventos/filtrar', [EventosController::class, 'filtrarFecha'])-
>name('eventos.filtrar');
```

```

//EXPORTAR EXCEL
Route::get('/eventos/{evento_id}/exportar-invitados',
[EventosController::class, 'exportarInvitados'])-
>name('eventos.exportarInvitados');

//RUTAS PARA REGISTRO FORMULARIO
Route::get('/eventos/{evento:public_id}/conductor/registro',
[EventosController::class, 'conductorForm'])->name('conductor.registro');

Route::post('/eventos/{evento:public_id}/conductor/registro/store',
[EventosController::class, 'conductorStore'])->name('conductor.registro.store');

```

Datos extras función exportarInvitados():

❖ Descripción general

Funciones para poder exportar el archivo en Excel:

```

class InvitadosExport implements FromCollection, WithHeadings
{
    protected $evento; //Protege la propiedad del evento.

    public function headings(): array
    {
        //Nombre del campos de las casillas.
        return [
            'Empresa',
            'CIF',
            'Nombre',
            'Apellidos',
            'Telefono',
            'Dni',
            'Email',
            'KAM',
            'Asistio'
        ];
    }

    public function __construct(Evento $evento)
    {
        $this->evento = $evento; //Guarda el evento para usarlo despues.
    }

    public function collection()
    {
        /**Recoge de cada evento el invitado y recorre con un bucle para que
         * me pinte todos los que hay ahí , ya que espera un array. */
        return $this->evento->invitados->map(function ($invitado) {
            return [
                'empresa'     => $invitado->empresa,
                'cif'          => $invitado->cif,
            ];
        });
    }
}

```

```

        'nombre'      => $invitado->nombre,
        'apellidos'   => $invitado->apellido,
        'telefono'    => "\t" . $invitado->telefono, //Se coloca "\t" para
que se veo el formato texto (numeros).
        'dni'         => "\t" . $invitado->dni, //Se coloca "\t" para que se
veo el formato texto (numeros).
        'email'       => $invitado->email,
        'kam'         => $invitado->kam,
        'asistio'     => $invitado->asiste ? 'Si' : 'No' //Condicion si el
invitado asiste o no asiste.
    ];
}
}
}

```

InvitadosController

Métodos principales

Método	Descripción
<code>index(Evento \$evento)</code>	Muestra una vista con la lista de invitados paginados.
<code>create(Evento \$evento)</code>	Muestra el formulario.
<code>store(Request \$request , \$id)</code>	Guarda un nuevo invitado en la base de datos.
<code>edit(Evento \$evento , Conductor \$invitado)</code>	Muestra el formulario para editar un invitado existente.
<code>update(Request \$request, Evento \$evento , Conductor \$invitado)</code>	Actualiza la información de un invitado.
<code>delete(\$id)</code>	Elimina un invitado.
<code>show(Request \$request , Evento \$evento)</code>	Busca por todos los campos en el buscador.
<code>actualizarAsistencia(Request \$request , Evento \$evento , Conductor \$invitado)</code>	Marca en el checkbox y guarda si el invitado asiste o no.
<code>importarInvitados(Request \$reuest , \$id)</code>	Importa la lista de invitados.

RUTAS InvitadosController

```

/**CRUD INVITADOS(CONDUCTORES) */
Route::prefix('eventos/{evento:id}')->group(function () {
    Route::get('invitados', [InvitadosController::class, 'index']-
>name('invitados.index');
    Route::get('invitados/create', [InvitadosController::class, 'create']-

```

```

>name('invitados.create');
    Route::post('invitados', [InvitadosController::class, 'store'])-
>name('invitados.store');

    Route::get('invitados/buscar', [InvitadosController::class, 'show'])-
>name('invitados.show');
    Route::get('invitados/{invitado}/edit', [InvitadosController::class,
'edit'])-
        ->name('invitados.edit');
    Route::put('invitados/{invitado}', [InvitadosController::class,
'update'])->name('invitados.update');
    Route::delete('invitados/delete', [InvitadosController::class, 'delete'])-
>name('invitados.delete');

    // Acciones específicas sobre el pivot
    Route::post(
        'invitados/{invitado}/asistencia',
        [InvitadosController::class, 'actualizarAsistencia']
    )->name('invitados.asistencia');

    // Import/Export ligados al evento> [InvitadosController::class,
'importarInvitados'])->name('invitados.importar');
    Route::get('invitados/exportar', [InvitadosController::class,
'exportarInvitados'])->name('invitados.exportar');
    Route::post('invitados/importar', [InvitadosController::class,
'importarInvitados'])->name('invitados.importar');
});

});

```

CochesController

❖ Métodos principales

Método	Descripción
index()	Muestra una vista con la lista de coches paginados.
create()	Muestra el formulario.
store()	Guarda un nuevo coche en la base de datos.
edit(\$id)	Muestra el formulario para editar un invitado existente.
update(Request \$request, \$id)	Actualiza la información de un invitado.
delete(\$id)	Elimina un invitado.
show(Request \$request)	Busca por todos los campos en el buscador.
actualizarAsistencia(\$id)	Marca en el checkbox y guarda si el coche tiene llave o no
exportarCoches(\$evento_id)	Exporta en Excel.
importarCoches(\$id)	Importa en Excel.

RUTAS CochesController

```
//CRUD COCHES
Route::get('/coches/{id}', [CochesController::class, 'index'])->name('coches.index');
Route::get('/coches/create/{id}', [CochesController::class, 'create'])->name('coches.create');
Route::post('/coches/{id}', [CochesController::class, 'store'])->name('coches.store');
Route::delete('/coches/delete/{id}', [CochesController::class, 'delete'])->name('coches.delete');
Route::get('/coches/edit/{id}', [CochesController::class, 'edit'])->name('coches.edit');
Route::put('/coches/update/{id}', [CochesController::class, 'update'])->name('coches.update');

Route::get('/coches/show/{id}', [CochesController::class, 'show'])->name('coches.show');
Route::post('/coches/{id}/actualizar', [CochesController::class, 'actualizarAsistencia']);

//EXPORTAR EXCEL
Route::get('/coches/{evento_id}/exportar-coches', [CochesController::class, 'exportarCoches'])->name('coches.exportarCoches');
//IMPORTAR EXCEL
Route::post('/coches/importar/{id}', [CochesController::class, 'importarCoches'])->name('coches.importarCoches');
```

TrazabilidadController

❖ Métodos principales

Método	Descripción
index(\$id)	Muestra la vista de toda la trazabilidad de coches (reservas)
show(Request \$request , \$id)	Buscador en toda la BD de la tabla trazabilidad
export()	Exporta la tabla trazabilidad

RUTAS TrazabilidadController

```
//TRAZABILIDAD DE PARADAS
Route::get('/trazabilidad/{evento}', [TrazabilidadController::class, 'index'])->name('trazabilidad.index');
Route::get('/trazabilidad/show/{id}', [TrazabilidadController::class, 'show'])->name('show.trazabilidad');
Route::get('/eventos/{evento}/reservas/export', [TrazabilidadController::class, 'export'])->name('reservas.export');
```

AjustesController

❖ Métodos principales

Método	Descripción
index(Request \$request , Evento \$evento)	Muestra la pagina entera con los datos cargados
storeParadas(Request \$request , Evento \$evento)	Crea parada
updateParadas(Request \$request , Evento \$evento , Parada \$parada)	Edita parada
deleteParadas(Evento \$evento , \$id)	Elimina parada
editCoches(Evento \$evento)	Recoge informacion y la usa para editar coche
updateCoches(Request \$request , Evento \$evento , Coch \$coche)	Edita coche
deleteCoches(Evento \$evento , \$id)	Elimina coche
storeRestaurante(Request \$request , Evento \$evento)	Crea restaurante
updateRestaurante(Request \$request , Evento \$evento , Restaurante \$restaurante)	Edita restaurante
deleteRestaurante(Evento \$evento , \$id)	Elimina restaurante
storeBanner(Request \$request , Evento \$evento)	Crea banner
updateBanner(Request \$request , Evento \$evento , Banner \$banner)	Edita banner
deleteBanner(Evento \$evento , \$id)	Elimina banner
storeTiming(Request \$request , Evento \$evento)	Crea timing
editTiming(Evento \$evento)	Recoge informacion para editar timing
updateTiming(Request \$request , Evento \$evento , Timing \$timing)	Edita timing
deleteTiming(Evento \$evento , \$id)	Elimina timing

RUTAS AjustesController

```

/***** RUTAS PARA AJUSTE (BOTON) */

// Mostrar página de ajustes de un evento concreto
Route::get('/ajustes/{evento}', [AjustesController::class, 'index'])
    ->name('admin.ajustes');

```

```

// CREA , EDITA y ELIMINA PARADA
Route::post('/ajustes/{evento}/paradas', [AjustesController::class,
'storeParadas'])->name('store.paradas');
Route::put('/ajustes/{evento}/paradas/{parada}', [AjustesController::class,
'updateParadas'])->name('evento.parada.update');
Route::delete('/ajustes/{evento}/paradas/delete/{id}', [AjustesController::class, 'deleteParadas'])
->name('eliminarParada');

//EDITA COCHES Y ELIMINA
Route::get('ajustes/{evento}/coches', [AjustesController::class,
'editCoches'])->name('evento.cochе.edit');
Route::put('ajustes/{evento}/coches/{coche}', [AjustesController::class,
'updateCoches'])->name('evento.cochе.update');
Route::delete('ajustes/{evento}/coches/delete/{id}', [AjustesController::class, 'deleteCoches'])
->name('eliminarCoches');

//CREA Y EDITA RESTAURANTE
Route::post('ajustes/{evento}/restaurante', [AjustesController::class,
'storeRestaurante'])->name('store.restaurantes');
Route::put('ajustes/{evento}/restaurante/{restaurante}', [AjustesController::class, 'updateRestaurante'])
->name('evento.restaurante.update');
Route::delete('ajustes/{evento}/restaurante/{id}', [AjustesController::class, 'deleteRestaurante'])
->name('eliminarRestaurante');

//CREA , EDITA Y ELIMINA BANNER
Route::post('ajustes/{evento}/banner', [AjustesController::class,
'storeBanner'])->name('store.banner');
Route::put('ajustes/{evento}/banner/{banner}', [AjustesController::class, 'updateBanner'])
->name('evento.banner.update');
Route::delete('ajustes/{evento}/banner/{id}', [AjustesController::class, 'deleteBanner'])
->name('eliminarBanner');

//CREA , EDITA Y ELIMINA TIMING
Route::get('ajustes/{evento}/edit', [AjustesController::class, 'editTiming'])
->name('evento.timing.edit');
Route::post('ajustes/{evento}/timing', [AjustesController::class, 'storeTiming'])
->name('store.timing');
Route::put('ajustes/{evento}/timing/{timing}', [AjustesController::class, 'updateTiming'])
->name('evento.timing.update');
Route::delete('ajustes/{evento}/timing/{id}', [AjustesController::class, 'deleteTiming'])
->name('eliminarTiming');

```

ClienteController

❖ Métodos principales

Método	Descripción
--------	-------------

Método	Descripción
index(Evento \$evento)	Muestra informacion de dashboard cliente
infoCoches(Evento \$evento)	Muestra informacion datos coches en vista cliente , todo con id
infoCochesAuto()	Muestra informacion coches , todo sin id
infoAauto()	Entrada a ruta , elige el ultimo evento
infoRestaurante(Evento \$evento)	Muestra informacion datos restaurante
principal()	Pagina principal dsahboard (cliente)

RUTAS ClienteController

```
//TRAZABILIDAD DE PARADAS
Route::get('/', [ClienteController::class, 'principal'])->name('inicio');
// Dashboard
Route::get('/cliente/dashboard', function () {
    return view('cliente.dashboard');
})->name('cliente.dashboard');

// ENTRADA a Ruta (elige el último evento y REDIRIGE a
/cliente/eventos/{evento})
Route::get('/cliente/ruta', [ClienteController::class, 'infoAuto'])
    ->name('cliente.ruta');

// Muestra la ruta de un evento concreto (renderiza la vista con $evento y
$paradas)
Route::get('/cliente/eventos/{evento}', [ClienteController::class, 'index'])
    ->name('cliente.eventos.show');

// Coches: entrada sin id (elige último y redirige)
Route::get('/cliente/coches', [ClienteController::class, 'infoCochesAuto'])
    ->name('cliente.info_coches');

// Coches: con id pinta la vista con datos
Route::get('/cliente/coches/{evento}', [ClienteController::class,
'infoCoches'])
    ->name('cliente.eventos.coches');

Route::get('/cliente/restaurante/{evento}', [ClienteController::class,
'infoRestaurante'])->name('cliente.info_restaurante');
```

PruebaDinamicaController

❖ Métodos principales

Método	Descripción
--------	-------------

Método	Descripción
cargaDatos()	Carga datos para coger informacion y pintar en el modal
storeReserva(Request \$request , Evento \$evento , Parada \$parada)	Crea la reserva

RUTAS ReservaController

```
//RUTAS RESERVA
Route::get('/cargarDatos', [ReservaController::class, 'cargaDatos'])->name('cargar.datos');
Route::post('/reserva/store/{evento}/{parada}', [ReservaController::class, 'storeReserva'])->name('store.reserva');
```

PruebaDinamicaController**❖ Métodos principales**

Método	Descripción
cargaDatos()	Carga datos para coger informacion y pintar en el modal
storePruebaDinamica(Request \$request)	Crea la prueba dinamica

RUTAS PruebaDinamicaController

```
//PRUEBA DINAMICA
Route::get('/cargarDatos/pruebaDinamica', [PruebaDinamicaController::class, 'cargaDatos'])->name('cargarDatos.pruebaDinamica');
Route::middleware('auth')->post('/store/pruebaDinamica', [PruebaDinamicaController::class, 'storePruebaDinamica'])->name('store.pruebaDinamica');
```

TimingController**❖ Métodos principales**

Método	Descripción
cargaDatos()	Carga datos para coger informacion y pintar en el modal

RUTAS TimingController

```
//TIMING
Route::get('/cargarDatos/timing', [TimingController::class, 'cargarDatos'])->name('cargarDatos.timing');
```

PatrocinadoresController

❖ Métodos principales

Método	Descripción
cargarDatos(Request \$request)	Carga datos para coger informacion y pintar en el modal

RUTAS PatrocinadoresController

```
//PATROCINADORES
Route::get('/cargarDatos/patrocinadores', [PatrocinadoresController::class, 'cargarDatos'])->name('cargarDatos.patrocinadores');
```

EventoConductorController

❖ Métodos principales

Método	Descripción
enviarEmail(\$evento_id , \$conductor_id)	Manda mensaje a correo electronico que se generara con un token y una url
mostrarFormulario(\$token)	Recupera datos del token y muestra formulario
enviarFormulario(Request \$request , \$token)	Envia el formulario con los datos

RUTAS EventoConductorController

```
/EVENTO CONDUCTOR
Route::get('/evento-confirmacion/{token}', [EventoConductorController::class, 'mostrarFormulario'])->name('evento.confirmacion');

//RUTA PARA ENVIAR FORMULARIO (PROCESADO).
Route::post('/evento-confirmacion/{token}', [EventoConductorController::class, 'enviarFormulario'])->name('evento.enviar');

//RUTA PARA ENVIAR EMAIL AL INVITADO.
Route::get('/evento/{evento_id}/invitado/{conductor_id}/confirmacion', [EventoConductorController::class, 'enviarEmail'])->name('invitados.enviarEmail');
```

🔗 Conclusión

Este proyecto ha sido desarrollado utilizando **Laravel 12** como framework principal del lado del servidor y **JavaScript** para las interacciones en el cliente. La estructura basada en el patrón **Modelo-Vista-Controlador (MVC)** permitió una clara separación de responsabilidades, facilitando tanto el desarrollo como el mantenimiento del sistema.

Gracias a Laravel, se logró una integración eficiente entre la lógica de negocio, la gestión de rutas y el acceso a la base de datos, mientras que JavaScript aportó dinamismo y una mejor experiencia para el usuario en la interfaz.

Logros principales:

- Implementación exitosa del patrón MVC con Laravel 12.
- Código organizado, reutilizable y fácil de escalar.
- Interfaz funcional y dinámica con JavaScript.

Recomendaciones para futuras mejoras:

- Incorporar autenticación basada en roles (admin, usuario, etc.).
- Agregar validaciones asíncronas del lado del cliente.
- Implementar pruebas automatizadas para asegurar la calidad del código.

Este proyecto representa una base sólida para continuar creciendo en complejidad y funcionalidad en futuras versiones.