**Mapa de archivos proyecto Laravel**

Estructura general de la página:

-Routes/web.php

-resources/views/layout.blade.php //Esta creo que es como una plantilla que usan las 4 siguientes

-resources/views/inicio.blade.php

-resources/views/menu.blade.php

-resources/views/reservas.blade.php

-resources/views/contacto.blade.php

**Para la sección de contacto:**

**0).env**

Para editar la configuración del email

**1) routes/web.php (Rutas)**

Define por dónde entra la petición (GET para ver el formulario, POST para enviarlo).

Mantendremos tu ruta GET y añadiremos una POST.

Opcional: en vez de un closure, usar un Controlador nos da orden y escalabilidad.

**2)app/Http/Controllers/ContactController.php (Controlador)**

Contiene la lógica cuando llega el POST: recibe datos, delega la validación (Form Request), y decide qué responder (redirigir con éxito o con errores).

**3)app/Http/Requests/StoreContactRequest.php (Form Request)**

Validación en servidor de los datos del formulario. Así, aunque alguien intente saltarse la validación del navegador, el backend sigue protegiendo.

**4)App/Mail/ContactMessage.php**

Define cómo se construye el email: asunto, vista, datos, etc

**5)Ressources/views/emails/contacto.blade.php**

Es la vista del email. Es decir, la plantilla con el mensaje que recibiré.

**6)resources/views/contacto.blade.php (Vista)**

Es la vista del formulario en la pagina web.

Añadimos validación del lado del cliente (HTML5: required, minlength, maxlength, type="email", pattern para el teléfono) y un poquito de JS (contador de caracteres, bloqueo de doble envío).

**Estructura de la sección de reservas**

**1) routes/web.php**

**•** Archivo de rutas web del proyecto.

• Define rutas para /, /menu, /reservas (GET y POST) y /contacto.

• Incluye validación de datos y creación de reservas

**2)App/Http/Controllers/ReservaController.php**

• Define la lógica de creación de la reserva

**3)app/Models/Reserva.php**

• Modelo Eloquent para la tabla reservas.

• Define los campos que pueden asignarse de forma masiva ($fillable).

**4)database/migrations/2025\_08\_164036\_create\_reservas\_table.php**

• Migración que crea la tabla reservas en la base de datos.

• Define columnas (fecha, hora, personas, etc.) y métodos up()/down().

**5)resources/views/reservas.blade.php**

• Vista Blade para el formulario de reserva.

• Muestramensajes de éxito y el formulario con selectores para fecha, personas y hora.

**6)App/Mail/ReservaConfirmada.php**

• Archivos de configuración para el envío automático del email:

**7)resources\views\emails/reserva\_confirmada.blade.php:**

la pantilla markdown del email

**Estructura de la sección menu:**

**M (Modelo)**

**1)app/Models/Plato.php**

* Representa la **entidad de dominio** “Plato”.
* Encapsula cómo hablamos con la BD: gracias a Eloquent puedes hacer Plato::create(), Plato::orderBy()->get(), etc.
* **Por qué**: así no escribes SQL a mano por todas partes y centralizas la lógica de datos.

**2)database/migrations/2025\_08\_16\_135645\_create\_platos\_table.php**

* Define la **forma** de la tabla platos (nombre, precio, imagen\_path…).
* **Por qué**: versionas la estructura de tu BD. Con migrate cualquier máquina crea la misma tabla igual.

**V (Vistas)**

**3)resources/views/layout.blade.php**

* Es tu **plantilla base** (layout). Solo estructura: <nav>, @yield('content').
* **Por qué**: evita duplicar HTML en todas las páginas.

**4)resources/views/menu.blade.php**

* Es la **vista concreta** del menú: recibe $platos ya preparados y los pinta (Blade: @foreach, asset('storage/...')).
* **Por qué**: la vista no consulta BD ni valida; solo **presenta** lo que le pasan.

**5)resources/views/platos/create.blade.php**

* Vista del **formulario** para crear platos (uso interno).
* **Por qué**: separa UI de creación/edición de la lógica que guarda.

**C (Controlador)**

**6)app/Http/Controllers/PlatoController.php**

* **index()**: pide datos al Modelo (Plato::…->get()), los **prepara** (agrupa por categoría) y los pasa a la Vista menu.
* **create()**: solo devuelve la vista del formulario.
* **store()**: recibe la Request, **valida**, guarda con el Modelo y, si hay imagen, la **sube** al disco public; después **redirecciona**.
* **Por qué**: el controlador orquesta el flujo HTTP → (Regla de negocio básica/validación) → Modelo → Vista/redirect.

**Enrutado (no es parte de MVC, pero es el “pegamento”)**

**7)routes/web.php**

* **Declara** qué controlador maneja cada URL:
  + GET /menu → PlatoController@index (listado público).
  + GET /platos/create → PlatoController@create (formulario).
  + POST /platos → PlatoController@store (guardar).
* **Por qué**: separa la capa HTTP (rutas) de la lógica (controladores). Es más legible y testeable.

**Flujo de petición (resumen mental)**

**Listar menú (GET /menu)**

1. **Ruta** GET /menu
2. **Controlador** index() → consulta Modelo Plato, agrupa.
3. **Vista** menu.blade.php → pinta nombre, precio, **imagen** con asset('storage/...').

**Crear plato (POST /platos)**

1. **Formulario** en platos/create envía multipart/form-data.
2. **Controlador** store():
   * Valida campos (reglas → por qué: seguridad e integridad).
   * Crea el **Plato** (Modelo/Eloquent).
   * Si hay imagen: store('platos','public') → guarda archivo en storage/app/public/platos/ y persiste la **ruta** (imagen\_path).
3. **Redirect** a platos.index (tu /menu) → se ve el plato nuevo.

**Dónde encaja storage:link**

* Crea el enlace **public/storage → storage/app/public**.
* **Por qué**: los ficheros subidos (no van en public/ directamente) se sirven de forma segura y estándar.
* En la Vista usas asset('storage/'.$plato->imagen\_path).

**Idea clave (el “por qué” de todo esto)**

* **Modelo**: habla con la BD y representa tus datos.
* **Vista**: solo pinta.
* **Controlador**: traduce una petición HTTP en acciones sobre modelos y respuestas (vistas/redirects).
* **Rutas**: mapa URL → Controlador.
* **Migraciones**: versionan la estructura de tu BD (infraestructura).
* **Storage**: separa archivos de la carpeta pública y los sirve correctamente.

\*Añadir platos

\*Hacer commit

\*Con esto ya estás trabajando auténtico MVC en Laravel. Si quieres, en el siguiente paso te enseño a refactorizar validaciones a un FormRequest (todavía más limpio) y a extraer la lógica de subida de imagen a un método del Modelo para dejar el controlador aún más fino. ¿Te apetece?