SISTEMATIZACIÓN DEL PROCESO DE PAUSAS ACTIVAS MEDIANTE EL DESARROLLO DE UNA APLICACIÓN WEB BASADO EN METODOLOGÍAS ÁGILES EN LA EMPRESA AUTONIKA S.A.S DE BOGOTA PARA EL 2025

MANUAL DE TÉCNICO

JULIAN DAVID RAMIREZ BORRAEZ

DEIBY STEVEN BERMEO HERNANDEZ

UNIVERSIDAD PILOTO DE COLOMBIA

SECCIONAL DEL ALTO MAGDALENA

FACULTAD DE INGENIERÍA

PROGRAMA DE INGENIERÍA DE SISTEMAS

GIRARDOT, CUNDINAMARCA

2025

CONTENIDO

pág.

LISTA DE TABLAS

pág.

LISTA DE FIGURAS

pág.

INTRODUCCIÓN

El propósito principal de este Manual Técnico es proporcionar una guía detallada sobre el desarrollo, despliegue y mantenimiento del sistema de gestión de pausas activas implementado en la empresa AUTONIKA S.A.S. Este documento está dirigido a desarrolladores, administradores de sistemas y personal técnico que participe en el ciclo de vida del sistema, ya sea para tareas de mejora, adaptación a nuevas sedes o mantenimiento correctivo.

Dado que el sistema busca automatizar y estructurar la práctica de pausas activas en entornos laborales, se hace especial énfasis en los aspectos técnicos necesarios para garantizar su funcionamiento óptimo: desde la configuración del entorno hasta la integración de módulos como la detección de posturas mediante inteligencia artificial, la generación de reportes y la administración segmentada por roles. Este manual también contempla consideraciones futuras, de forma que facilite la continuidad y evolución del sistema en nuevas versiones o adaptaciones institucionales.

1 ALCANCE DE LA APLICACIÓN

La aplicación web desarrollada permite gestionar de forma automatizada el proceso de pausas activas dentro del entorno laboral, integrando funcionalidades como la detección de posturas mediante cámara, la asignación de ejercicios físicos y cognitivos, la retroalimentación por voz y el registro automático de cada actividad.

El sistema opera con tres roles definidos: Empleado, Talento Humano (RRHH) y Administrador, considerados suficientes para cubrir las necesidades funcionales del proyecto, simplificando la gestión y asegurando un control segmentado y eficaz.

Los usuarios solo podrán realizar los ejercicios físicos previamente definidos en el sistema, ya que cada uno fue codificado individualmente, lo cual implicó un proceso técnico detallado y demandante durante el desarrollo.

La aplicación está diseñada para ser escalable, permitiendo su despliegue en futuras sedes o en otras organizaciones con estructuras similares, sin modificar su núcleo funcional.

2 TECNOLOGÍAS UTILIZADAS

La aplicación web para la gestión de pausas activas se desarrolla utilizando tecnologías modernas que garantizan la eficiencia y seguridad. Entre las principales tecnologías utilizadas se encuentran:

Lenguaje de programación:

- **PHP 8.2:** Lenguaje de programación principal utilizado para el desarrollo del backend de la aplicación.

Framework y herramientas del ecosistema de Laravel:

- **Laravel 12:** Framework PHP que organiza el desarrollo del backend con estructuras MVC, migraciones, rutas, etc.
- **Jetstream:** Starter kit oficial de Laravel que facilita la autenticación, gestión de sesiones y paneles de usuario.
- **Liveware:** Librería para crear interfaces dinámicas en Laravel sin usar JavaScript directamente (se integra con Blade).
- **Vite:** Herramienta de construcción (build tool) para compilar y servir recursos frontend como CSS y JS rápidamente.

Base de datos:

- **MySQL 8.0:** Sistema de gestión de bases de datos relacional (usualmente MariaDB en entornos similares).

Entorno de desarrollo:

- **Visual Studio Code (VS Code):** Editor de código fuente con extensiones para facilitar el desarrollo con PHP y Laravel.

Librerías / Paquetes (Instalados por Composer):

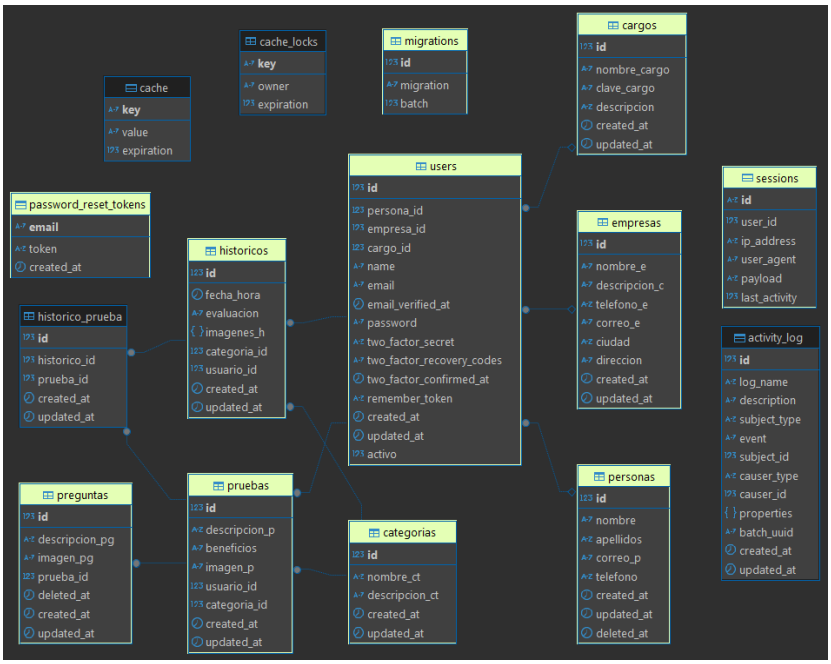
- **Spatie/Permission:** Manejo de roles y permisos en Laravel.

- Maatwebsite/Excel: Importación y exportación de archivos Excel y CSV.
- **Laravel DomPDF:** Generación de archivos PDF desde vistas Blade.
- **Blade Heroicons:** Conjunto de íconos SVG para usar fácilmente en Blade.
- **Spatie/Activitylog:** Registra automáticamente acciones del usuario en el sistema.

2.1 ESTRUCTURA DE LA BASE DE DATOS

La base de datos fue diseñada siguiendo un modelo relacional que garantiza integridad referencial y eficiencia en las operaciones CRUD del sistema. Incluye entidades como usuarios, empresas, documentos y prácticas, interconectadas mediante claves foráneas que reflejan las relaciones funcionales entre los datos. La estructura se visualiza a través de un diagrama ER que facilita su comprensión y mantenimiento.

Figura 1. Diagrama ER



Nota: Elaboración por los autores a partir de DBeaver, 2024

3 REQUISITOS DEL SISTEMA

La aplicación web para la gestión de pausas activas, desarrollada en Laravel 12 con Livewire y Jetstream, requiere recursos moderados para su funcionamiento eficiente en entornos de desarrollo.

3.1 REQUISITOS DEL HARDWARE

Se recomienda el siguiente hardware mínimo:

- **Procesador**: Doble núcleo a 2.4 GHz o superior.
- **Memoria RAM**: 8 GB o más, especialmente al compilar con Vite y usar simultáneamente servicios como cola y servidor.
- **Almacenamiento**: Al menos 30 GB disponibles para archivos del sistema, base de datos, dependencias y backups de prueba.
- **Otros**: Conectividad estable a internet para instalar dependencias y ejecutar comandos remotos (Composer, NPM).

Estos requisitos pueden incrementarse en un entorno de producción o si se trabaja con múltiples instancias del sistema.

3.2 REQUISITOS DEL SOFTWARE

La aplicación web para la gestión de pausas activas es compatible con diversos entornos. Se requiere el siguiente software para su correcta instalación y ejecución:

• **Sistema Operativo**: Linux, Windows o macOS

• **Servidor Web**: Apache o Nginx

• **Base de Datos:** MariaDB 10.5 o superior (compatible también con MySQL 8.0)

• **PHP**: Versión 8.2 o superior

• **Composer:** Última versión estable

• **Node.js y NPM**: Requeridos para la gestión de dependencias del frontend y el uso de Vite

• **Editor de Código (opcional para desarrollo):** Visual Studio Code con extensiones para PHP y Laravel

• **Navegador Web**: Compatible con las últimas versiones de Chrome, Firefox, Edge o Safari

Estos elementos son necesarios tanto para entornos de desarrollo como para su despliegue en producción.

3.3 CONFIGURACIÓN RECOMENDADA

Para garantizar un rendimiento óptimo y una mayor seguridad en el entorno de producción, se recomienda aplicar las siguientes configuraciones en el servidor donde se despliega la aplicación:

3.3.1 Habilitación de la extensión PHP OPCache

La extensión OPCache mejora significativamente el rendimiento al almacenar en caché los scripts PHP compilados, reduciendo el tiempo de carga de la aplicación. Esta opción puede activarse desde el panel de control del hosting (cPanel) en la sección de configuración de PHP.

3.3.2 Uso del motor PHP-FPM

Configurar el servidor web para utilizar PHP-FPM (FastCGI Process Manager) permite una gestión eficiente de los procesos PHP, lo cual mejora la capacidad de respuesta y la estabilidad de la aplicación. En Hostinger, esta opción se encuentra disponible y debe estar activada por defecto.

3.3.3 Configuración de HTTPS mediante certificado SSL

Se recomienda asegurar todas las conexiones mediante HTTPS, utilizando un certificado SSL. Hostinger ofrece certificados gratuitos a través de Let's Encrypt, los cuales pueden activarse fácilmente desde el panel de control. Esto protege la información transmitida y evita vulnerabilidades comunes asociadas a conexiones inseguras.

Estas configuraciones, aunque opcionales, son altamente recomendadas para entornos en producción, ya que fortalecen tanto el rendimiento como la seguridad general del sistema.

4 ANÁLISIS Y DESEÑO FUNCIONAL

4.1 HISTORIAS DE USUARIO

Estas historias guiaron el diseño, desarrollo e implementación de las funcionalidades del sistema, asegurando que cada módulo responda a un propósito concreto dentro del flujo operativo de la organización.

Tabla 1. HU01

<table>
<tr><td>Historia de usuario#1</td><td colspan="3">Iniciar sesión en la plataforma</td></tr>
<tr><td>Como</td><td colspan="3">Empleado</td></tr>
<tr><td>Quiero</td><td colspan="3">iniciar sesión con mis credenciales</td></tr>
<tr><td>Para poder</td><td colspan="3">acceder a mis rutinas, historial y reportes de pausas activas.</td></tr>
<tr><td colspan="2" rowspan="3">Validación :

- El sistema debe validar el usuario y contraseña.
- Debe mostrar mensajes de error si los datos son incorrectos.
- El acceso debe estar restringido según rol (empleado / administrador).</td><td>Valor:</td><td>300</td></tr>
<tr><td>Prioridad :</td><td>1</td></tr>
<tr><td>Estimación:</td><td>4h.</td></tr>
</table>

Fuente: Elaboración por los autores a partir de Excel, 2024

Tabla 2. HU02

<table>
<tr><td>Historia de usuario#2</td><td colspan="3">Ejecutar rutina de pausa activa</td></tr>
<tr><td>Como</td><td colspan="3">Empleado</td></tr>
<tr><td>Quiero</td><td colspan="3">realizar una rutina de ejercicios de pausa activa</td></tr>
<tr><td>Para poder</td><td colspan="3">mejorar mi salud física y mental durante la jornada laboral.</td></tr>
<tr><td colspan="2" rowspan="3">Validación :

- El sistema debe mostrar una rutina automática con ejercicios secuenciales.
- Debe incluir imágenes de referencia y temporizador.
- La rutina debe concluir al terminar todos los ejercicios.</td><td>Valor:</td><td>250</td></tr>
<tr><td>Prioridad :</td><td>2</td></tr>
<tr><td>Estimación:</td><td>6h.</td></tr>
</table>

Fuente: Elaboración por los autores a partir de Excel, 2024

Tabla 3. HU03

Historia de usuario#3	**Validar postura mediante cámara**		
Como	Empleado		
Quiero	que el sistema valide que realizo correctamente cada postura		
Para poder	asegurarme de que el ejercicio cuenta solo si se hace bien.		
Validación : - El sistema debe usar la cámara para detectar la postura. - La repetición cuenta solo si la postura se mantiene correctamente. - Debe avisar mediante voz si la postura fue válida o no.		**Valor:**	280
		Prioridad :	3
		Estimación:	12h.

Fuente: Elaboración por los autores a partir de Excel, 2024

Tabla 4. HU04

Historia de usuario#4	**Visualizar historial de pausas activas**		
Como	Empleado		
Quiero	consultar mi historial de pausas activas		
Para poder	conocer cuántas he realizado y cuándo.		
Validación : - El historial debe mostrar fecha, duración y tipo de ejercicios. - Debe estar disponible desde el perfil del usuario. - Mostrar como lista o tabla exportable.		**Valor:**	200
		Prioridad :	4
		Estimación:	5h.

Fuente: Elaboración por los autores a partir de Excel, 2024

Tabla 5. HU05

Historia de usuario#5	**Descargar reporte en PDF**		
Como	Recursos humanos (RRHH)		
Quiero	generar un reporte en PDF con las pausas activas realizadas		
Para poder	llevar un respaldo de la actividad en el sistema.		
Validación : - El reporte debe incluir nombre, fechas, ejercicios realizados. - Debe permitir descarga directa. - Diseño legible		**Valor:**	180
		Prioridad :	5
		Estimación:	7h.

Fuente: Elaboración por los autores a partir de Excel, 2024

Tabla 6. HU06

Historia de usuario #6	**Administrar usuarios**		
Como	Administrador		
Quiero	gestionar cuentas de usuario		
Para poder	controlar el acceso a la plataforma.		
Validación : - Crear, editar o desactivar usuarios. - Asignar roles (empleado o administrador). - Mostrar lista de usuarios activos.		**Valor:**	150
		Prioridad :	6
		Estimación:	6h.

Fuente: Elaboración por los autores a partir de Excel, 2024

Tabla 7. HU07

Historia de usuario #7	**Gestionar ejercicios cognitivos**		
Como	Recursos humanos (RRHH)		
Quiero	agregar o modificar ejercicios cognitivos		
Para poder	mantener actualizada la base mental.		
Validación : - Solo modificar ejercicios cognitivos. - Agregar título, descripción e imagen. - Ver cambios reflejados en la plataforma.		**Valor:**	160
		Prioridad :	7
		Estimación:	5h.

Fuente: Elaboración por los autores a partir de Excel, 2024

Tabla 8. HU08

Historia de usuario #8	**Seguridad y protección de datos**		
Como	Administrador		
Quiero	garantizar la protección de datos		
Para poder	cumplir con la normativa y mantener la privacidad.		
Validación : - Uso de contraseñas cifradas. - Accesos limitados por rol. - Protección frente a accesos no autorizados.		**Valor:**	220
		Prioridad :	2
		Estimación:	8h.

Fuente: Elaboración por los autores a partir de Excel, 2024

Tabla 9. HU09

Historia de usuario #9	**Visualizar reportes por empleado**		
Como	Recursos humanos (RRHH)		
Quiero	ver reportes de pausas activas por usuario		
Para poder	hacer seguimiento al cumplimiento del programa.		
Validación : - Debe permitir buscar por nombre o fecha. - El reporte debe ser visualmente claro. - Debe mostrarse la prueba del ejercicio (captura).		**Valor:**	180
		Prioridad :	5
		Estimación:	6h.

Fuente: Elaboración por los autores a partir de Excel, 2024

Tabla 10. HU10

Historia de usuario #10	**Ver instrucciones con imágenes de referencia**		
Como	Empleado		
Quiero	ver cómo se realiza cada ejercicio con imágenes		
Para poder	asegurarme de ejecutar correctamente cada postura.		
Validación : - Cada ejercicio debe tener imagen clara y nombre visible. - Debe mostrarse en pantalla antes de iniciar el conteo. - Debe ser clara y concisa la instrucción.		**Valor:**	200
		Prioridad :	4
		Estimación:	4h.

Fuente: Elaboración por los autores a partir de Excel, 2024

Tabla 11. HU11

Historia de usuario #11	**Contador automático de repeticiones**		
Como	Empleado		
Quiero	que el sistema lleve la cuenta de mis repeticiones		
Para poder	enfocarme en realizar los movimientos sin distraerme.		
Validación : - Mostrar contador en pantalla con cada repetición. - Reproducir audio al completar cada repetición y al finalizar la serie. - Reiniciar automáticamente en el siguiente ejercicio.		**Valor:**	250
		Prioridad :	3
		Estimación:	10h.

Fuente: Elaboración por los autores a partir de Excel, 2024

Tabla 12. HU12

Historia de usuario #12	**Instrucciones por síntesis de voz**	
Como	Empleado	
Quiero	recibir indicaciones por voz durante la rutina	
Para poder	seguir las instrucciones sin tener que mirar la pantalla.	
Validación : - Anunciar inicio del ejercicio, tiempo restante y finalización. - Usar una voz clara y pausada. - Presente en todos los ejercicios físicos	**Valor:**	220
	Prioridad :	3
	Estimación:	6h.

Fuente: Elaboración por los autores a partir de Excel, 2024

Tabla 13. HU13

Historia de usuario #13	**Registro automático de actividad**	
Como	Empleado	
Quiero	que cada rutina completada se registre automáticamente	
Para poder	que no tenga que ingresar nada manualmente.	
Validación : - Registrar fecha, hora, ejercicios realizados y duración. - Guardar en base de datos del usuario. - Permitir exportar o consultar más tarde.	**Valor:**	200
	Prioridad :	4
	Estimación:	5h.

Fuente: Elaboración por los autores a partir de Excel, 2024

Tabla 14. HU14

Historia de usuario #14	**Auditoría de actividades en el panel administrativo**	
Como	Administrador	
Quiero	mirar toda la actividad realizada en el panel administrativo	
Para poder	hacer seguimiento de la actividad de los usuarios.	
Validación : - Incluir nombre del usuario, fechas, id. - Mostrar módulos afectados o actividades realizadas. - Permitir exportar o consultar más tarde.	**Valor:**	200
	Prioridad :	4
	Estimación:	5h.

Fuente: Elaboración por los autores a partir de Excel, 2024

4.2 CASO DE USO

Esta sección complementa las historias de usuario al describir de manera estructurada cómo cada usuario logra sus objetivos dentro del sistema, definiendo el flujo normal, las excepciones y las condiciones asociadas a cada proceso.

4.2.1 Definiciones de caso de uso

Las definiciones de casos de uso permiten detallar paso a paso cómo interactúan los actores con el sistema para lograr un objetivo específico.

Tabla 15. Descripción de casos de uso / Iniciar rutina de pausa activa

Caso de uso	Iniciar rutina de pausa activa
Actor principal	Empleado
Descripción	Permite al empleado iniciar una rutina de ejercicios de pausa activa desde la aplicación.
Precondiciones	El empleado debe haber iniciado sesión. Debe estar dentro de su horario laboral.
Flujo de eventos principal	El empleado escoge la rutina que quiere realizar. El sistema le muestra una explicación breve de como realizar la rutina. Inicia el primer ejercicio con nombre e imagen. El sistema comienza el contador de repeticiones.
Flujo de eventos alternativos	Si el empleado utiliza la opción aleatoria se iniciar una rutina al azar. Si no hay conexión a internet, no se puede trabajar.
Postcondiciones	La rutina queda marcada como iniciada y en ejecución.
Requisitos asociados	HU01, HU02, HU03

Fuente: Elaboración por los autores a partir de Excel, 2024

Tabla 16. Descripción de casos de uso / Detectar y validar postura del ejercicio

Caso de uso	Detectar y validar postura del ejercicio
Actor principal	Sistema
Descripción	Permite validar que el empleado está realizando correctamente la postura del ejercicio.
Precondiciones	La cámara debe estar activada y funcionando. El ejercicio debe estar en ejecución.

Flujo de eventos principal	El sistema analiza el esqueleto del usuario usando MoveNet. Compara la postura detectada con la esperada. Si es correcta, inicia el conteo de tiempo o repeticiones. Si se completa correctamente, se registra la repetición.
Flujo de eventos alternativos	Si la postura no es válida, se muestra retroalimentación y se repite la detección. Si no se detecta al usuario, se cancela el ejercicio.
Postcondiciones	El sistema registra la ejecución de la postura como válida.
Requisitos asociados	HU03, HU04

Fuente: Elaboración por los autores a partir de Excel, 2024

Tabla 17. Descripción de casos de uso / Finalizar rutina de pausa activa

Caso de uso	Finalizar rutina de pausa activa
Actor principal	Sistema
Descripción	Permite completar una rutina y registrar los resultados.
Precondiciones	Deben haberse realizado todos los ejercicios de la rutina. El sistema debe haber validado todas las posturas.
Flujo de eventos principal	El sistema detecta la finalización del último ejercicio. Calcula duración total y repeticiones. Registra la información en la base de datos. Muestra mensaje de rutina completada.
Flujo de eventos alternativos	Si la rutina se interrumpe, no se guarda a menos de que complete toda la rutina.
Postcondiciones	La rutina queda registrada con fecha, hora y datos de ejecución.
Requisitos asociados	HU03, HU06

Fuente: Elaboración por los autores a partir de Excel, 2024

Tabla 18. Descripción de casos de uso / Generar y descargar reporte en PDF

Caso de uso	Generar y descargar reporte en PDF
Actor principal	Administrador o Recursos humanos (RRHH)
Descripción	Permite generar un reporte de rutinas completadas y descargarlo en PDF.
Precondiciones	El usuario debe haber completado al menos una rutina. Debe tener permisos para acceder al historial.
Flujo de eventos principal	El usuario selecciona un rango de fechas. El sistema consulta las rutinas registradas. Ofrece la opción de descargar o visualizar.
Flujo de eventos alternativos	Si no hay rutinas en ese período, se informa al usuario.

Postcondiciones	El reporte queda generado y/o descargado.
Requisitos asociados	HU05, HU11

Fuente: Elaboración por los autores a partir de Excel, 2024

Tabla 19. Descripción de casos de uso / Registrar ejecución de rutina

Caso de uso	Registrar ejecución de rutina
Actor principal	Sistema
Descripción	Guarda en la base de datos cada rutina completada por el empleado, con su información detallada.
Precondiciones	El ejercicio debe haberse realizado y validado correctamente. El usuario debe estar identificado.
Flujo de eventos principal	Al completar cada ejercicio, se almacena nombre, pruebas(capturas), fecha y hora. Al finalizar toda la rutina, se relaciona con el usuario. Se vincula con la prueba o sesión correspondiente.
Flujo de eventos alternativos	Si se pierde conexión, no se realiza el registro.
Postcondiciones	La ejecución queda registrada para futuros reportes y análisis.
Requisitos asociados	HU04, HU06, HU10

Fuente: Elaboración por los autores a partir de Excel, 2024

Tabla 20. Descripción de casos de uso / Administrar usuarios del sistema

Caso de uso	Administrar usuarios del sistema
Actor principal	Administrador o Recursos humanos (RRHH)
Descripción	Permite al administrador crear, modificar o eliminar usuarios.
Precondiciones	El administrador debe haber iniciado sesión.
Flujo de eventos principal	El administrador accede al panel de usuarios. Selecciona crear, editar o eliminar un usuario. El sistema valida y guarda los cambios.
Flujo de eventos alternativos	Si falta información, se muestra error y se solicita completar.
Postcondiciones	La base de datos de usuarios queda actualizada.
Requisitos asociados	HU07

Fuente: Elaboración por los autores a partir de Excel, 2024

Tabla 21. Descripción de casos de uso / Consultar historial de rutinas

Caso de uso	Consultar historial de rutinas
Actor principal	Empleado o RRHH

Descripción	Permite revisar las rutinas realizadas por un empleado en un período determinado.
Precondiciones	El usuario debe tener permisos para consultar.
Flujo de eventos principal	El usuario selecciona fechas y nombre del empleado (si aplica). El sistema muestra la lista de rutinas completadas.
Flujo de eventos alternativos	Si no hay registros, se muestra mensaje.
Postcondiciones	Se visualiza el historial con detalles por rutina.
Requisitos asociados	HU06, HU08

Fuente: Elaboración por los autores a partir de Excel, 2024

Tabla 22. Descripción de casos de uso / Gestionar ejercicios disponibles

Caso de uso	Gestionar ejercicios disponibles
Actor principal	Administrador o Recursos humanos (RRHH)
Descripción	Permite agregar, modificar o quitar ejercicios del sistema (solo categoría cognitivos).
Precondiciones	El administrador o RRHH debe estar autenticado.
Flujo de eventos principal	Accede al panel de ejercicios. Visualiza la lista actual. Crea, edita o elimina ejercicios.
Flujo de eventos alternativos	Si hay ejercicios en uso, se bloquea la eliminación.
Postcondiciones	El catálogo de ejercicios queda actualizado.
Requisitos asociados	HU12

Fuente: Elaboración por los autores a partir de Excel, 2024

Tabla 23. Descripción de casos de uso / Ejecutar rutina con asistencia por voz

Caso de uso	Ejecutar rutina con asistencia por voz
Actor principal	Sistema
Descripción	Guía al usuario durante la rutina mediante mensajes por voz.
Precondiciones	La rutina debe estar en ejecución.
Flujo de eventos principal	Al iniciar el ejercicio, se anuncia el nombre. Se indica el tiempo o repeticiones restantes. Se confirma por voz la finalización del ejercicio.
Flujo de eventos alternativos	Si el usuario silencia el sonido, se muestra texto alternativo.
Postcondiciones	El usuario completa la rutina con retroalimentación auditiva.
Requisitos asociados	HU09

Fuente: Elaboración por los autores a partir de Excel, 2024

Tabla 24. Descripción de casos de uso / Realizar rutina de ejercicios cognitivos

Caso de uso	Realizar rutina de ejercicios cognitivos
Actor principal	Empleado
Descripción	Permite al empleado realizar una rutina de ejercicios para activar funciones cognitivas.
Precondiciones	El empleado debe haber iniciado sesión. Debe estar habilitada la opción de ejercicios cognitivos.
Flujo de eventos principal	El sistema presenta un ejercicio cognitivo (ej. Sopa de letras). El empleado interactúa con el reto. El sistema evalúa y guarda el resultado.
Flujo de eventos alternativos	Si no se completa, se descarta el resultado.
Postcondiciones	El resultado queda almacenado para análisis.
Requisitos asociados	HU13

Fuente: Elaboración por los autores a partir de Excel, 2024

Tabla 25. Descripción de casos de uso / Controlar acceso seguro al sistema

Caso de uso	Controlar acceso seguro al sistema
Actor principal	Todos los usuarios
Descripción	Permite acceder al sistema de forma autenticada y segura.
Precondiciones	El usuario debe estar registrado.
Flujo de eventos principal	El usuario accede a la pantalla de login. Ingresa sus credenciales. El sistema valida e inicia la sesión.
Flujo de eventos alternativos	Si las credenciales son incorrectas, se muestra un error. Si el usuario está deshabilitado, no se permite el acceso.
Postcondiciones	El usuario tiene acceso a sus funcionalidades según rol (cargo).
Requisitos asociados	HU07, HU14

Fuente: Elaboración por los autores a partir de Excel, 2024

Tabla 26. Descripción de casos de uso / Insertar captura de postura en reporte

Caso de uso	Insertar captura de postura en reporte
Actor principal	Sistema
Descripción	Incluye automáticamente una imagen de la postura en el histórico generado.
Precondiciones	Debe haberse validado al menos una postura correctamente.
Flujo de eventos principal	El sistema toma una captura al validar la postura. La guarda en el sistema. Al generar el histórico, se inserta en la sección correspondiente.

Flujo de eventos alternativos	Si la cámara falla, no se puede realizar el ejercicio físico.
Postcondiciones	El reporte incluye evidencia visual de la ejecución.
Requisitos asociados	HU10

Fuente: Elaboración por los autores a partir de Excel, 2024

Tabla 27. Descripción de casos de uso / Relacionar ejecución con prueba específica

Caso de uso	Relacionar ejecución con prueba específica
Actor principal	Sistema
Descripción	Vincula cada ejecución de rutina con una sesión de prueba concreta para seguimiento.
Precondiciones	Debe existir una prueba o sesión activa.
Flujo de eventos principal	El sistema identifica la sesión de prueba en curso. Registra cada ejercicio ejecutado dentro de esa prueba. Asocia resultados y capturas a la sesión.
Flujo de eventos alternativos	Si no hay sesión activa, no se puede guardar la ejecución.
Postcondiciones	Los datos quedan relacionados para reportes personalizados.
Requisitos asociados	HU11

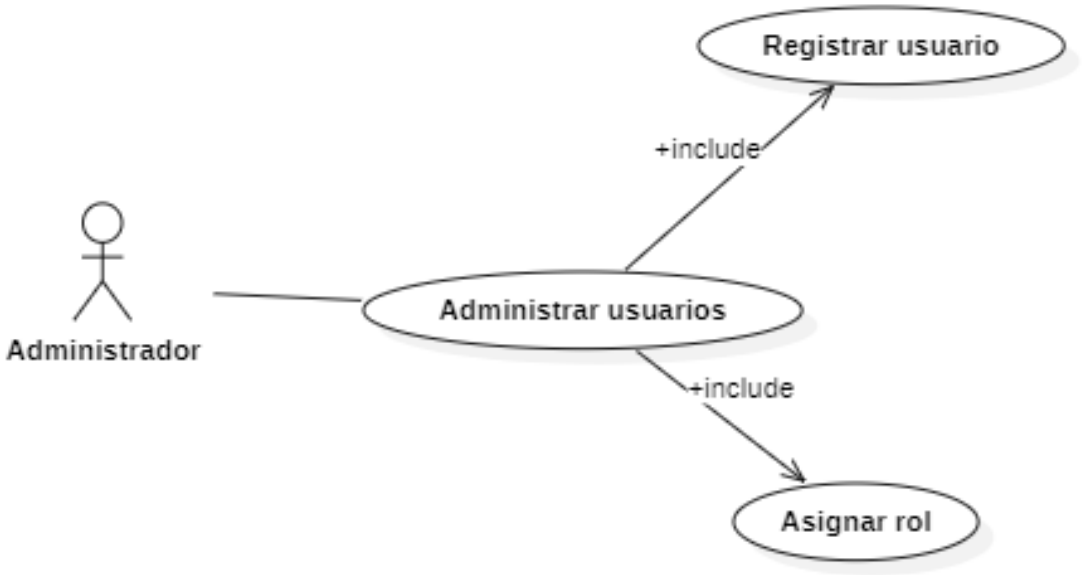
Fuente: Elaboración por los autores a partir de Excel, 2024

4.2.2 Diagramas de caso de uso

A través de estos diagramas se identifican y representan de manera visual los principales procesos que conforman el funcionamiento del sistema. Estos esquemas permiten comprender con mayor claridad cómo interactúan los distintos componentes de la plataforma, incluyendo las funcionalidades disponibles, los flujos de trabajo internos y los roles de usuario involucrados en cada operación. Además, facilitan la identificación de las entradas, salidas y decisiones que se toman en cada etapa, lo cual resulta fundamental para el análisis, la documentación y la mejora continua del sistema.

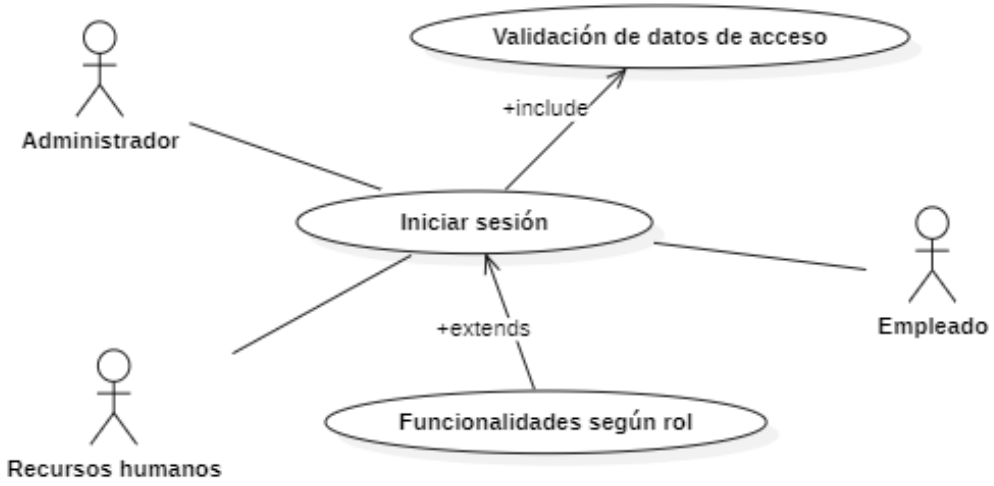
Los diagramas también cumplen una función clave en la comunicación entre los equipos de desarrollo, soporte y usuarios finales, ya que proporcionan una visión estructurada de los procesos, reduciendo ambigüedades y facilitando la capacitación.

Figura 2. Caso de uso / Registrar usuario



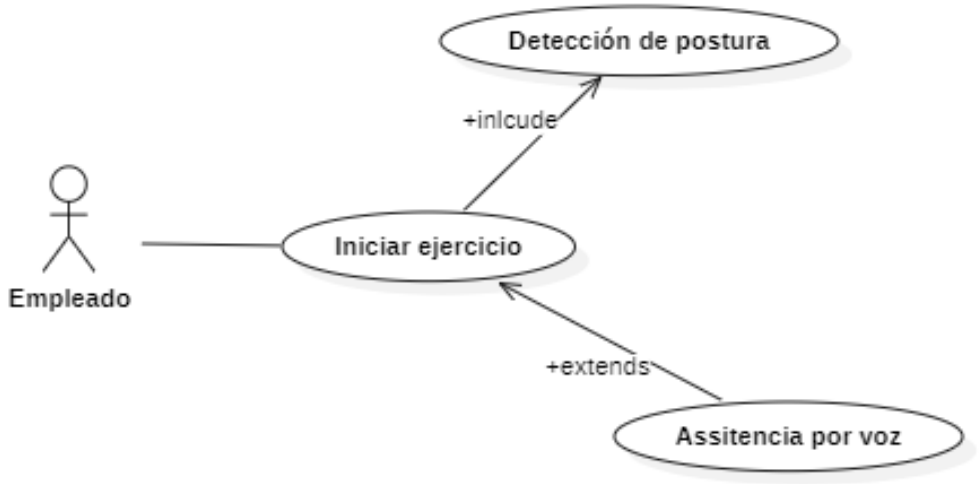
Fuente: Elaboración por los autores a partir de Excel, 2024

Figura 3. Caso de uso / Iniciar sesión



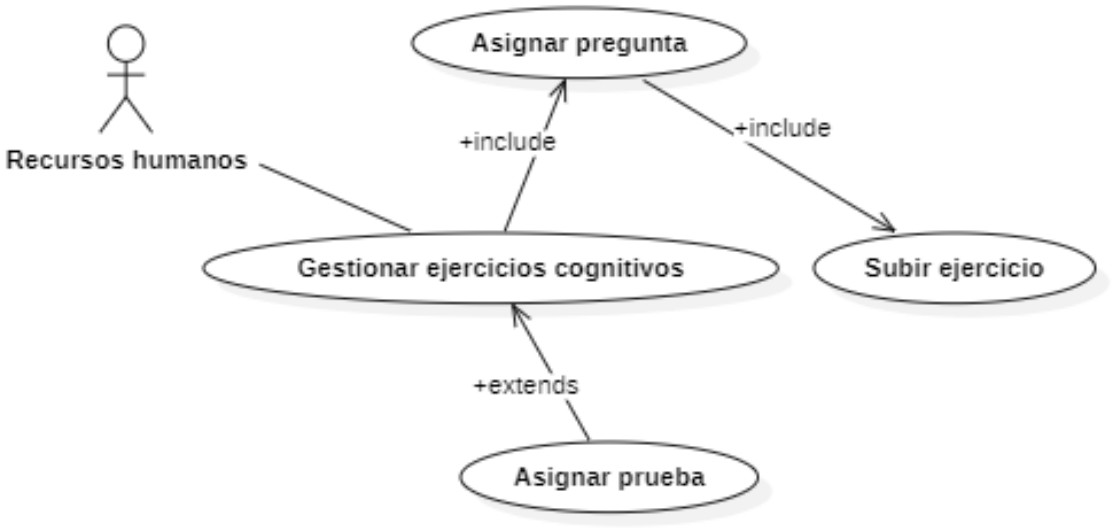
Fuente: Elaboración por los autores a partir de Excel, 2024

Figura 4. Caso de uso / Iniciar ejercicio



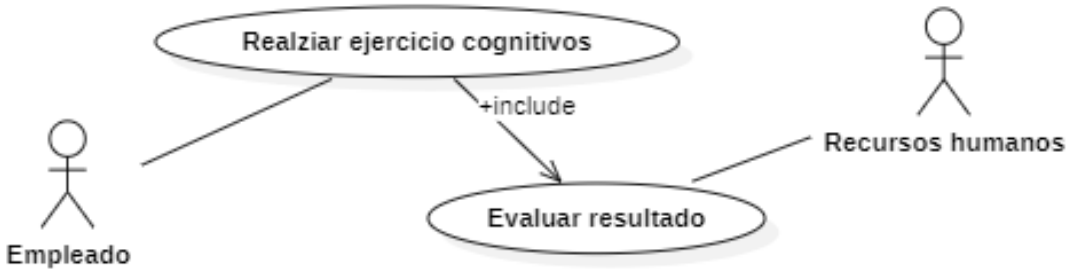
Fuente: Elaboración por los autores a partir de Excel, 2024

Figura 5. Caso de uso / Gestión de ejercicios cognitivos



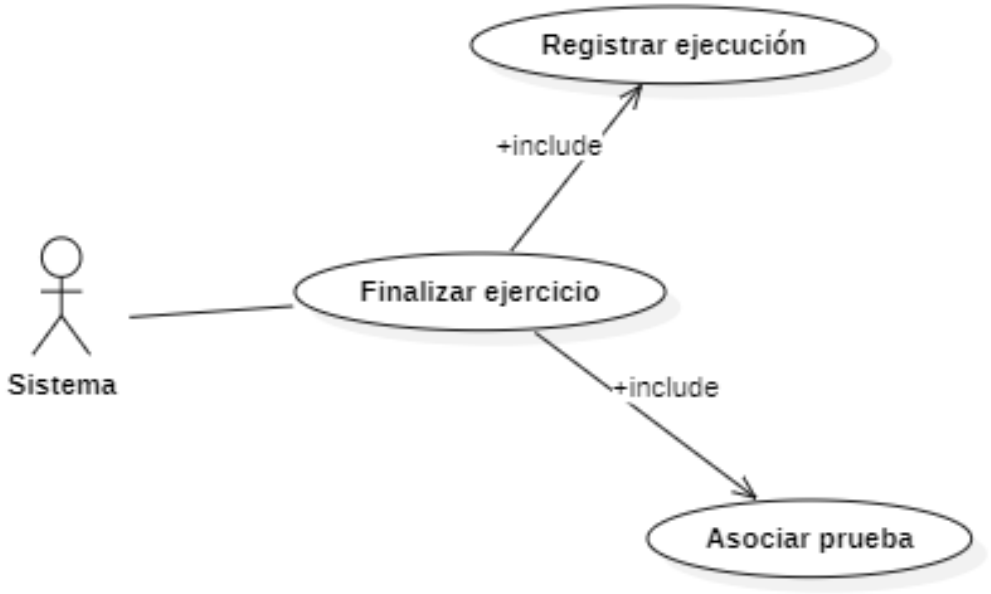
Fuente: Elaboración por los autores a partir de Excel, 2024

Figura 6. Caso de uso / Realización de ejercicio cognitivo



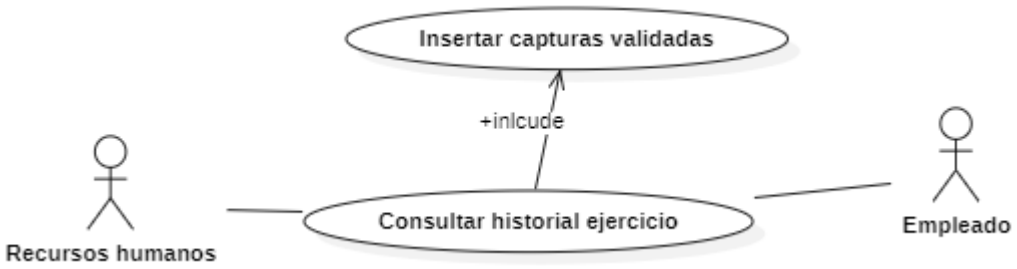
Fuente: Elaboración por los autores a partir de Excel, 2024

Figura 7. Caso de uso / Finalizar ejercicio



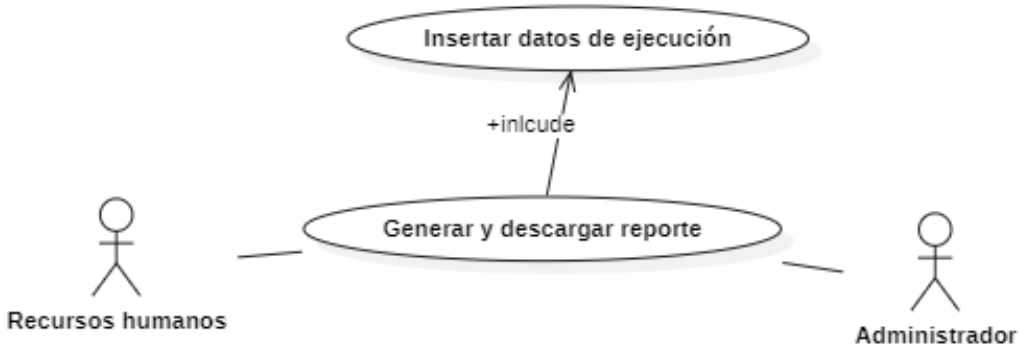
Fuente: Elaboración por los autores a partir de Excel, 2024

Figura 8. Caso de uso / Consultar históricos



Fuente: Elaboración por los autores a partir de Excel, 2024

Figura 9. Caso de uso / Generar reporte



Fuente: Elaboración por los autores a partir de Excel, 2024

4.3 DICCIONARIO DE DATOS

Esta sección proporciona una referencia técnica fundamental para desarrolladores, administradores y cualquier miembro del equipo que requiera comprender la estructura interna de la información manejada por la aplicación.

Tabla 28. Empresas

Columna	Tipo de dato	Descripción
id	int	ID de la empresa
nombre_e	varchar	Nombre de la empresa
descripcion_c	text	Descripción de la empresa
telefono_e	varchar	Teléfono de contacto
correo_e	varchar	Correo electrónico
ciudad	varchar	Ciudad
direccion	text	Dirección física
created_at	datetime	Fecha de creación
updated_at	datetime	Fecha de actualización

Fuente: Elaboración por los autores a partir de Excel, 2024

Tabla 29. Personas

Columna	Tipo de dato	Descripción
id	int	ID de la persona
nombre	varchar	Nombre
apellidos	varchar	Apellidos
correo_p	varchar	Correo electrónico
telefono	varchar	Teléfono
created_at	datetime	Fecha de creación
updated_at	datetime	Fecha de actualización
deleted_at	datetime	Fecha de eliminación lógica

Fuente: Elaboración por los autores a partir de Excel, 2024

Tabla 30. Sesiones

Columna	Tipo de dato	Descripción
id	varchar	ID de sesión
user_id	int	ID del usuario
ip_address	varchar	Dirección IP del usuario
user_agent	text	Navegador / dispositivo del usuario
payload	text	Datos de la sesión
last_activity	datetime	Última actividad registrada

Fuente: Elaboración por los autores a partir de Excel, 2024

Tabla 31. Cargos

Columna	Tipo de dato	Descripción
id	int	ID del cargo
nombre_cargo	varchar	Nombre del cargo
clave_cargo	varchar	Clave interna del cargo
descripcion	text	Descripción del cargo
created_at	datetime	Fecha de creación

Fuente: Elaboración por los autores a partir de Excel, 2024

Tabla 32. Usuarios

Columna	Tipo de dato	Descripción
id	int	ID del usuario
persona_id	int	ID de la persona asociada
empresa_id	int	ID de la empresa
cargo_id	int	ID del cargo (rol)
name	varchar	Nombre de usuario
email	varchar	Correo electrónico
email_verified_at	datetime	Fecha de verificación del correo
password	varchar	Contraseña encriptada
two_factor_secret	varchar	Secreto para autenticación 2FA
two_factor_recovery_codes	text	Códigos de recuperación 2FA
two_factor_confirmed_at	datetime	Confirmación de 2FA
remember_token	varchar	Token de sesión persistente
created_at	datetime	Fecha de creación
updated_at	datetime	Fecha de actualización
activo	boolean	Estado activo/inactivo

Fuente: Elaboración por los autores a partir de Excel, 2024

Tabla 33. Categorías

Columna	Tipo de dato	Descripción
id	int	ID de la categoría
nombre_ct	varchar	Nombre de la categoría
descripcion_ct	text	Descripción de la categoría
created_at	datetime	Fecha de creación
updated_at	datetime	Fecha de actualización

Fuente: Elaboración por los autores a partir de Excel, 2024

Tabla 34. Pruebas

Columna	Tipo de dato	Descripción
id	int	ID de la prueba
imagen_p	varchar	Ruta de imagen asociada
descripcion_p	text	Descripción de la prueba
beneficios	text	Beneficios esperados
usuario_id	int	ID del usuario que crea la prueba
categoria_id	int	ID de la categoría
created_at	datetime	Fecha de creación
updated_at	datetime	Fecha de actualización

Fuente: Elaboración por los autores a partir de Excel, 2024

Tabla 35. Preguntas

Columna	Tipo de dato	Descripción
id	int	ID de la pregunta
descripcion_pg	text	Texto de la pregunta
imagen_pg	varchar	Ruta de imagen asociada
prueba_id	int	ID de la prueba
deleted_at	datetime	Fecha de eliminación lógica
created_at	datetime	Fecha de creación
updated_at	datetime	Fecha de actualización

Fuente: Elaboración por los autores a partir de Excel, 2024

Tabla 36. Históricos

Columna	Tipo de dato	Descripción
id	int	ID del histórico
fecha_hora	datetime	Fecha y hora del evento
evaluacion	text	Resultado de la evaluación
imagenes_h	json	Imágenes relacionadas
categoria_id	int	ID de categoría asociada
usuario_id	int	ID del usuario
created_at	datetime	Fecha de creación
updated_at	datetime	Fecha de actualización

Fuente: Elaboración por los autores a partir de Excel, 2024

Tabla 37. Histórico de pruebas

Columna	Tipo de dato	Descripción
id	int	ID del histórico de prueba
historico_id	int	ID del histórico
prueba_id	int	ID de la prueba
created_at	datetime	Fecha de creación
updated_at	datetime	Fecha de actualización

Fuente: Elaboración por los autores a partir de Excel, 2024

Tabla 38. Log de actividades

Columna	Tipo de dato	Descripción
id	int	ID del log de actividad
log_name	varchar	Nombre del log
description	text	Descripción de la actividad
subject_type	varchar	Tipo de objeto afectado (modelo)
subject_id	int	ID del objeto afectado
event	varchar	Tipo de evento (crear, actualizar...)
subject	json	Contenido del objeto afectado
causer_type	varchar	Tipo de ejecutor
causer_id	int	ID del ejecutor
properties	json	Propiedades adicionales del evento
batch_uuid	varchar	UUID de lote
created_at	datetime	Fecha de creación del evento
updated_at	datetime	Fecha de actualización del evento

Fuente: Elaboración por los autores a partir de Excel, 2024

5 PROCESO DE INSTALACIÓN EN EL HOSTING

Antes de cargar el proyecto Laravel al hosting, se deben realizar algunos ajustes importantes en la configuración del entorno para asegurar el correcto funcionamiento en producción:

Primero se debe realizar un ajuste del entorno en el archivo .env, en la raíz del proyecto Laravel, se debe configurar el entorno de producción modificando las siguientes variables en el archivo .env:

```
APP_ENV=production
APP_DEBUG=false
```

Estas variables desactivan el modo debug y activan configuraciones optimizadas para un entorno seguro y estable. Después se realiza la división del proyecto para despliegue compartido, dado que muchos servicios de hosting compartido (como Hostinger) no permiten personalizar el directorio raíz, es necesario separar el proyecto en dos partes para proteger la estructura del framework:

- Comprimir en un archivo ZIP todas las carpetas del proyecto excepto la carpeta public/.
- Comprimir en otro archivo ZIP solo el contenido de la carpeta public/.
- Ambos archivos se subirán a ubicaciones específicas como se indica en el siguiente paso.

5.1 CONFIGURACIÓN EN EL HOSTING

Primero verificar la versión de PHP

Acceder al panel de administración del hosting (por ejemplo, hPanel o cPanel) y asegurarse de que esté configurada la versión de PHP 8.2 o superior, ya que es requerida por Laravel 12. Esto puede configurarse desde la sección “Configuración de PHP” del panel.

Segundo subir los archivos del proyecto

- En el "Administrador de Archivos", crear una carpeta llamada laravel (o un nombre similar) en la raíz del servidor.
- Subir y descomprimir allí el archivo ZIP que contiene todas las carpetas del proyecto excepto public/.
- Luego, acceder a la carpeta public_html y subir allí el ZIP con el contenido de public/. Descomprimir directamente en public_html, de modo que index.php y los archivos de frontend estén accesibles públicamente.

5.2 CONFIGURACIÓN DE LA APLICACIÓN EN EL HOSTING

Una vez que los archivos han sido cargados y descomprimidos correctamente en sus respectivas ubicaciones, es necesario realizar algunos ajustes adicionales para que Laravel funcione correctamente desde el entorno del hosting compartido.

Primero realiza la modificación del archivo index.php

En la carpeta public_html, abrir el archivo index.php y modificar las siguientes líneas:

```
require __DIR__.'/../laravel/vendor/autoload.php';
$app = require_once __DIR__.'/../laravel/bootstrap/app.php';
```

Reemplaza laravel por el nombre de la carpeta que contiene el framework (es decir, donde descomprimiste el primer archivo ZIP). Este cambio le indica a Laravel dónde se encuentra su estructura principal.

Posteriormente redefine la ruta pública del proyecto

Para que Laravel reconozca correctamente el directorio público en el entorno del hosting, se debe modificar el archivo AppServiceProvider.php ubicado en app/Providers/. Dentro del método register(), se debe agregar el siguiente código:

```
public function register()
{
    $this->app->bind('path.public', function () {
        return '/home/usuario/public_html';
    });
}
```

Asegúrate de reemplazar usuario con el nombre real del usuario asignado por el hosting, el cual puedes verificar en la ruta completa del archivo dentro del Administrador de Archivos (por ejemplo, /home/miusuario/public_html).

Estos ajustes aseguran que la aplicación pueda ejecutarse correctamente desde el entorno del hosting, incluso cuando la estructura original de Laravel ha sido modificada por restricciones del proveedor.

5.3 IMPLEMENTACIÓN DE LA BASE DE DATOS EN EL SERVICIO DE HOSTING

Para que la aplicación pueda acceder correctamente a la base de datos desde el servidor de producción, se deben seguir los siguientes pasos desde el panel de control del servicio de hosting (por ejemplo, hPanel de Hostinger):

Primer paso: Crear una Base de Datos

Desde la sección "Bases de datos MySQL" del panel de control, se procede a crear una nueva base de datos. En este paso, se debe definir:

- Nombre de la base de datos.
- Nombre de usuario asociado a la base de datos.
- Contraseña segura para el usuario.

Es importante guardar esta información, ya que se usará posteriormente en la configuración de Laravel.

Segundo paso: Importar la Estructura de la Base de Datos

- Ingresar a phpMyAdmin desde el panel de control del hosting.
- Seleccionar la base de datos recién creada.
- Hacer clic en la pestaña Importar.
- Subir el archivo .sql que contiene la estructura y/o datos iniciales de la base de datos.
- Hacer clic en Continuar para ejecutar la importación.

Tercer paso: Configuración de Usuarios y Privilegios

Verifica que el usuario asignado a la base de datos tenga todos los privilegios habilitados especialmente los necesarios para realizar operaciones como (SELECT, INSERT, UPDATE, DELETE, etc.). Esto puede gestionarse desde la sección de privilegios o gestión de usuarios del panel.

Cuarto paso: Configuración en el Archivo .env

En el archivo .env ubicado en la raíz del proyecto Laravel (ya cargado al hosting), se deben ajustar las variables de conexión a la base de datos con los datos creados en el primer paso:

```
DB_CONNECTION=mysql
DB_HOST=localhost
DB_PORT=3306
DB_DATABASE=nombre_de_la_base_de_datos
DB_USERNAME=nombre_de_usuario
DB_PASSWORD=contraseña_de_usuario
```

Nota: En la mayoría de los servicios compartidos, DB_HOST puede permanecer como localhost. Si el proveedor asigna un host diferente, debe reemplazarse por el valor indicado en el panel.

6 DESARROLLO

6.1 MIGRACIONES

Después de completar la configuración inicial del entorno y del hosting, el desarrollo de la aplicación web se centra en la construcción de funcionalidades específicas utilizando el ecosistema de Laravel 12. A continuación, se describen aspectos clave que deben tenerse en cuenta durante el proceso de desarrollo:

6.1.1 Definición de migraciones

Las migraciones en Laravel 12 permiten gestionar la estructura de la base de datos de forma controlada y versionada, facilitando tanto la creación como la modificación de tablas a lo largo del ciclo de vida del proyecto. Estas se almacenan en el directorio database/migrations y se ejecutan mediante comandos Artisan.

Creación de migraciones

Para generar una nueva migración, se utiliza el siguiente comando de Artisan:

```
php artisan make:migration nombre_de_la_migracion
```

Este comando creará un archivo dentro del directorio database/migrations, con un nombre basado en la convención de Laravel que incluye la fecha y hora de creación, garantizando un orden cronológico.

Estructura del archivo de migración

Cada archivo de migración contiene dos métodos principales:

- **up():** Define los cambios que se deben aplicar a la base de datos, como la creación de tablas, columnas, índices o relaciones.
- **down():** Indica cómo revertir los cambios realizados por el método up().

Migración básica:

```
public function up()
{
    Schema::create('nombre_de_la_tabla', function (Blueprint
$table) {
        $table->id();
        $table->string('columna');
        $table->timestamps();
    });
}

public function down()
{
    Schema::dropIfExists('nombre_de_la_tabla');
}
```

Estas migraciones se aplican mediante el comando php artisan migrate, y pueden revertirse con php artisan migrate:rollback.

6.1.2 Columnas y tipos de datos

Dentro del método up() de una migración, se definen las columnas de la tabla utilizando el objeto $table. Laravel proporciona una variedad de métodos para declarar distintos tipos de datos compatibles con MySQL/MariaDB.

A continuación, se presentan algunos casos comunes:

```
Schema::create('usuarios', function (Blueprint $table) {
    $table->id(); // Clave primaria autoincremental
    $table->string('nombre'); // Cadena de texto (VARCHAR)
    $table->integer('edad'); // Número entero
    $table->boolean('activo'); // Valor booleano (true/false)
    $table->timestamps(); // Crea las columnas created_at y
updated_at
});
```

Laravel también permite definir otros tipos de columnas como:

- $table->text('descripcion'); → para textos largos.
- $table->date('fecha_nacimiento'); → para fechas.
- $table->float('precio', 8, 2); → para valores decimales.
- $table->foreignId('rol_id')->constrained(); → para llaves foráneas.

Estas definiciones permiten estructurar correctamente la base de datos de acuerdo con las necesidades del sistema.

6.1.3 Restricción y modificaciones

En Laravel, además de definir el tipo de dato de cada columna, es posible aplicar restricciones y modificadores para controlar su comportamiento dentro de la base de datos. Estas se agregan mediante métodos encadenados al definir cada columna.

Algunas de las restricciones y modificadores más comunes son:

```
Schema::create('usuarios', function (Blueprint $table) {
    $table->id(); // Clave primaria por defecto
    $table->string('email')->unique(); // Valor único
    $table->string('nombre')->default('Invitado'); // Valor por
defecto
    $table->boolean('activo')->default(true); // Valor por
defecto para booleano
    $table->timestamps();
});
```

También es posible definir claves primarias e índices manualmente:

```
$table->primary('id'); // Clave primaria personalizada
$table->index('nombre'); // Índice para mejorar búsqueda
```

Estos modificadores permiten garantizar la integridad de los datos y optimizar el rendimiento de las consultas.

6.1.4 Ejecución de migraciones

Una vez que se han definido correctamente las migraciones en el directorio database/migrations, se pueden aplicar los cambios a la base de datos mediante el comando Artisan:

```
php artisan migrate
```

Este comando ejecuta todas las migraciones pendientes y crea las tablas, columnas y restricciones definidas. Laravel registra automáticamente qué migraciones ya han sido ejecutadas para evitar repeticiones.

6.1.5 Revertir migraciones

En caso de que se necesite deshacer los cambios realizados por una migración, Laravel ofrece el comando:

```
php artisan migrate:rollback
```

Este comando ejecuta el método down() de las últimas migraciones ejecutadas, eliminando las tablas o modificaciones que estas hayan aplicado.

Si se desea revertir varias migraciones, se puede especificar el número de pasos:

```
php artisan migrate:rollback --step=2
```

Y si se requiere eliminar todas las migraciones y volver a aplicar desde cero:

```
php artisan migrate:fresh
```

Estas opciones son útiles durante el desarrollo para ajustar la estructura de la base de datos sin tener que modificarla manualmente.

6.1.6 Estado de migraciones

Laravel mantiene un registro del estado de cada migración ejecutada en la tabla migrations, creada automáticamente en la base de datos. Esta tabla almacena el nombre del archivo de migración y su marca de tiempo, lo cual permite identificar qué migraciones han sido aplicadas y evitar su ejecución duplicada.

6.2 MODELOS

Los modelos en Laravel representan las entidades de la base de datos y encapsulan la lógica de acceso a sus datos. Cada modelo está asociado a una tabla específica, permitiendo trabajar con sus registros como si fueran objetos PHP, a través del ORM Eloquent.

Laravel facilita la creación de modelos mediante el comando Artisan:

```
php artisan make:model NombreDelModelo
```

Este comando genera un archivo en el directorio app/Models, donde se puede definir la lógica relacionada con dicha entidad, incluyendo relaciones con otras tablas, atributos accesibles, castings y más.

6.2.1 Definición de modelos

Un modelo en Laravel es una clase PHP que extiende la clase base Illuminate\Database\Eloquent\Model. Representa una tabla en la base de datos y permite interactuar con sus registros de forma orientada a objetos mediante el ORM Eloquent.

A continuación, un caso básico de la definición de un modelo llamado Usuario, asociado a una tabla usuarios:

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Usuario extends Model
{
    // Definición del modelo
}
```

Laravel asocia automáticamente el modelo con la tabla usuarios, siguiendo la convención de pluralizar el nombre del modelo en minúsculas.

6.2.2 Eloquent ORM

Laravel utiliza Eloquent, su sistema ORM (Object-Relational Mapping), para facilitar la interacción con la base de datos mediante modelos. Eloquent permite realizar operaciones comunes como crear, consultar, actualizar y eliminar registros (CRUD) de forma intuitiva y orientada a objetos.

6.2.3 Tabla asociada

Por convención, Eloquent asume que el nombre de la tabla asociada a un modelo es el plural en minúsculas del nombre de la clase del modelo. Por ejemplo, un modelo llamado Empleado se vincularía automáticamente con la tabla empleados.

Sin embargo, cuando la tabla no sigue esta convención, se puede definir manualmente utilizando la propiedad protegida $table dentro del modelo:

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Empleado extends Model
{
    protected $table = 'nombre_de_la_tabla';
}
```

Esta flexibilidad permite trabajar con bases de datos existentes o con estructuras personalizadas sin modificar el esquema actual.

6.2.4 Relaciones Eloquent

Laravel Eloquent proporciona distintos tipos de relaciones que permiten establecer conexiones lógicas entre modelos, facilitando la manipulación de datos relacionados. A continuación, se describen las más comunes:

Uno a Uno (One to One)

Esta relación se usa cuando un modelo tiene exactamente un registro relacionado.

```
public function perfil()
{
    return $this->hasOne(Perfil::class);
}
```

Uno a Muchos (One to Many)

Se utiliza cuando un modelo tiene múltiples registros relacionados.

```
public function comentarios()
{
    return $this->hasMany(Comentario::class);
}
```

Muchos a Muchos (Many to Many)

Permite que ambos modelos tengan múltiples relaciones entre sí. Se requiere una tabla intermedia.

```
public function roles()
{
    return $this->belongsToMany(Rol::class);
}
```

Relación Polimórfica

Este tipo de relación permite que un modelo pertenezca a más de un otro modelo mediante una sola asociación.

```
public function imagen()
{
    return $this->morphOne(Imagen::class, 'imageable');
}
```

6.2.5 Claves foráneas locales

Al establecer relaciones entre modelos en Laravel utilizando Eloquent, es fundamental definir correctamente las claves foráneas y locales que conectan las tablas. Laravel sigue convenciones automáticas: la clave foránea se asume como el nombre del modelo en singular seguido de _id, y la clave local suele ser id. Sin embargo, cuando se trabaja con claves personalizadas o nombres de columna distintos, se deben definir explícitamente.

Ejemplo:

```
public function ejercicios()
{
    return $this->hasMany(Ejercicio::class, 'usuario_id',
'codigo_usuario');
}
```

En el ejemplo 'usuario_id' es la clave foránea en la tabla ejercicios y ‘codigo_usuario' es la clave local en la tabla usuarios. Esto permite que Eloquent gestione correctamente la relación, incluso si los nombres de columna no siguen las convenciones estándar.

6.2.6 Acceso a relaciones

En Laravel, las relaciones entre modelos se acceden utilizando los métodos definidos en el modelo correspondiente. Por ejemplo, si un modelo Usuario tiene una relación "uno a muchos" con un modelo Post, se puede acceder a los posts del usuario de la siguiente forma:

```
$usuario = Usuario::find(1);
$posts = $usuario->posts;
```

posts es el nombre del método definido en el modelo Usuario que representa la relación. Laravel carga automáticamente la colección de objetos relacionados.

6.2.7 Eager Loading

Para optimizar el rendimiento al recuperar modelos con relaciones, se puede utilizar Eager Loading. Este enfoque reduce la cantidad de consultas ejecutadas, cargando las relaciones de los modelos en una sola consulta o en consultas agrupadas.

En lugar de realizar múltiples consultas para obtener los posts de cada usuario, podemos utilizar el método with() para cargar las relaciones de manera eficiente.

```
$usuarios = Usuario::with('posts')->get();
```

En este caso, se cargan todos los usuarios junto con sus respectivos posts en una consulta optimizada. Laravel realizará dos consultas: una para obtener los usuarios y otra para obtener los posts relacionados, evitando el problema de N+1 consultas.

Eager Loading con múltiples relaciones

Si necesitas cargar varias relaciones al mismo tiempo, puedes hacerlo de esta manera:

```
$usuarios = Usuario::with(['posts', 'comentarios'])->get();
```

Esto carga tanto los posts como los comentarios relacionados con cada usuario en dos consultas optimizadas.

6.2.8 Políticas de acceso

Laravel ofrece un sistema robusto para controlar el acceso a los modelos mediante Políticas de Acceso. Estas políticas permiten definir quién puede realizar ciertas acciones en los modelos, como crear, editar o eliminar registros.

Implementación

Autenticación y Roles: Laravel utiliza la interfaz Authorizable y el trait HasRoles de Spatie para gestionar roles y permisos. Al utilizar HasRoles, podemos asignar roles a los usuarios y verificar sus permisos de manera sencilla.

Modelo Usuario con roles:

```
use Illuminate\Database\Eloquent\Model;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Spatie\Permission\Traits\HasRoles;

class Usuario extends Authenticatable
{
    use HasRoles;
}
```

Definir Políticas: Las políticas permiten definir métodos para autorizar acciones específicas, como verificar si un usuario puede crear o actualizar un modelo. Laravel facilita la creación de políticas con el comando:

```
php artisan make:policy NombreDeLaPolitica --model=Modelo
```

Esto genera una política con métodos para cada acción que se puede realizar sobre el modelo, como create, update, delete.

Aplicar Políticas:

Luego, en el controlador o en las rutas, puedes aplicar la política utilizando el método authorize:

```
public function update($id)
{
    $usuario = Usuario::findOrFail($id);
    $this->authorize('update', $usuario);
}
```

De esta manera, antes de proceder con una acción, Laravel verifica si el usuario tiene permiso para realizarla.

El uso de políticas de acceso, junto con el manejo de relaciones y roles, permite un diseño de base de datos flexible, modular y fácil de mantener, ya que las reglas de acceso se separan claramente de la lógica de negocio.

6.3 CONTROLADORES

Los controladores en Laravel manejan la lógica de la aplicación y actúan como intermediarios entre las rutas y los modelos. Permiten centralizar operaciones comunes como la creación, edición y eliminación de registros. En el contexto de la aplicación web de pausas activas, una estructura clara de controladores garantiza un desarrollo escalable, ordenado y fácil de mantener. Además, se integran de forma natural con funcionalidades modernas como Livewire y la gestión de usuarios mediante Jetstream.

6.3.1 Definición de controladores

Un controlador en Laravel es una clase PHP ubicada en el directorio app/Http/Controllers. Esta clase extiende de la clase base Controller y se encarga de recibir solicitudes HTTP (Request), procesarlas, y devolver una respuesta apropiada, ya sea una vista, un JSON o una redirección.

Para generar un nuevo controlador, se utiliza el siguiente comando de Artisan:

```
php artisan make:controller NombreControlador
```

Este comando creará un archivo dentro del directorio Http/Controllers, con el nombre que definas en el comando.

Estructura de un controlador básico:

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class MiControlador extends Controller
{
    public function index()
    {
        // Lógica de la acción
    }
}
```

Laravel también permite generar controladores con estructura de recurso (--resource) para facilitar la implementación CRUD. Esto resulta útil para manejar entidades como usuarios, pruebas, ejercicios o reportes dentro de la aplicación.

6.3.2 Métodos en controladores

Los métodos dentro de un controlador representan acciones concretas que responden a rutas definidas en la aplicación web. Cada método puede encargarse de tareas específicas como mostrar una vista, procesar un formulario, crear registros o interactuar con componentes Livewire.

En Laravel, estos métodos son invocados por las rutas configuradas en los archivos de rutas (web.php, api.php, etc.), y pueden recibir parámetros desde la URL o desde las solicitudes HTTP.

Método para mostrar el perfil de un usuario:

```
public function mostrarPerfil($id)
{
    // Lógica para mostrar el perfil del usuario con el ID
proporcionado
    $usuario = User::findOrFail($id);
    return view('usuarios.perfil', compact('usuario'));
}
```

Además, en controladores tipo recurso, Laravel define automáticamente métodos como index, create, store, show, edit, update y destroy, permitiendo mantener una estructura estándar en la aplicación.

6.3.3 Inyección de dependencias

Laravel permite utilizar la inyección de dependencias en los controladores, lo que mejora la claridad del código y fomenta una arquitectura desacoplada. Esto facilita el acceso a clases como Request, modelos personalizados o servicios, sin necesidad de instanciarlos manualmente. Cuando Laravel detecta que un método requiere una instancia específica, automáticamente la resuelve desde el contenedor de servicios. Ejemplo de inyección del objeto Request:

```
use Illuminate\Http\Request;

public function index(Request $request)
{
    // Acceso al objeto Request inyectado automáticamente
    $filtros = $request->query('filtro');
    // Lógica con base en los filtros...
}
```

Este mecanismo también es útil para inyectar repositorios, servicios personalizados o incluso componentes Livewire, alineándose con las prácticas modernas de desarrollo en Laravel 12.

6.3.4 Respuestas del controlador

Los controladores en Laravel pueden devolver distintos tipos de respuestas dependiendo del flujo de la aplicación web. Estas pueden ser vistas Blade, datos en formato JSON, redirecciones, entre otras, facilitando así una interacción flexible entre el servidor y el cliente.

Laravel proporciona métodos expresivos como view(), json(), redirect() y back() para manejar estos escenarios:

```
// Retornar una vista
public function mostrarVista()
{
    return view('nombre_de_la_vista');
}

// Retornar una respuesta JSON
public function obtenerDatos()
{
    return response()->json(['estado' => 'ok']);
}

// Redireccionar a otra ruta
public function guardar()
{
    // lógica...
    return redirect()->route('ruta.destino');
}
```

Este enfoque permite que la aplicación web responda de manera adecuada según el contexto de uso, tanto en entornos tradicionales como en componentes Livewire o API.

6.3.5 Middleware en controladores

Los middleware en Laravel permiten interceptar las solicitudes HTTP antes o después de que lleguen al controlador. Se pueden aplicar directamente en el constructor del controlador para asegurar, por ejemplo, que el usuario esté autenticado o tenga ciertos permisos antes de acceder a una acción.

Esto es especialmente útil para controlar el acceso a ciertas secciones de la aplicación web, validar roles o aplicar filtros globales.

```
public function __construct()
{
    $this->middleware('auth');
    $this->middleware('role:admin')->only('index');
}
```

En este caso, el middleware auth asegura que el usuario esté autenticado para acceder a cualquier método del controlador, y role:admin restringe el acceso al método index solo a usuarios con el rol de administrador.

6.3.6 Rutas y controladores

En Laravel, las rutas se definen en el archivo routes/web.php y se asocian a métodos específicos dentro de los controladores. Este enfoque mejora la organización del código y permite mantener una estructura clara y modular, facilitando la gestión de las rutas y su lógica correspondiente.

Por ejemplo, para asociar una ruta a un controlador, puedes definirla de la siguiente manera:

```
Route::get('/perfil/{id}', [MiControlador::class,
'mostrarPerfil']);
```

La ruta GET /perfil/{id} es manejada por el método mostrarPerfil del controlador MiControlador. La sintaxis moderna utiliza el nombre completo del controlador con ::class, una buena práctica para evitar errores de cadena de texto y mejorar la legibilidad.

6.3.7 Validación de datos

En Laravel, la validación de datos se realiza fácilmente dentro de los controladores, utilizando el objeto Request. Esto asegura que los datos recibidos en una solicitud cumplan con los criterios establecidos antes de ser procesados o almacenados en la base de datos.

A continuación, se muestra un ejemplo de cómo validar los datos al actualizar el perfil de un usuario:

```
public function actualizarPerfil(Request $request)
{
    // Validación y actualización del perfil
    $request->validate([
        'nombre' => 'required|string|max:255',
        'email' => 'required|email|unique:usuarios,email,' .
$request->user()->id,
    ]);

    $request->user()->update($request->only('nombre', 'email'));

    return redirect()->route('perfil')->with('success', 'Perfil
actualizado correctamente.');
}
```

En este ejemplo, la función actualizarPerfil valida y actualiza de manera compacta los datos del perfil del usuario:

- **Validación de los datos**: Primero, se valida que el campo nombre sea obligatorio, una cadena de texto, y que no supere los 255 caracteres. El campo email también es obligatorio, debe ser un correo válido, y debe ser único en la base de datos, excepto para el usuario que está realizando la solicitud.
- **Actualización de los datos**: Después de la validación, se usa el método update() para actualizar de manera eficiente los campos nombre y email del usuario. El método only() se asegura de que solo se actualicen estos campos.
- **Redirección y mensaje de éxito**: Finalmente, la función redirige a la ruta perfil con un mensaje de éxito que confirma que el perfil ha sido actualizado correctamente.

Mensaje de error

Si la validación falla, Laravel automáticamente redirigirá al usuario de vuelta con los errores correspondientes, los cuales podrán ser mostrados en la vista. Un ejemplo de cómo manejar estos errores en tu vista sería:

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

Este bloque de código en la vista muestra los errores de validación, si es que existen, asegurando que el usuario sea informado.

6.3.8 Controladores de recursos

Laravel proporciona controladores de recursos para simplificar la implementación de operaciones CRUD (Crear, Leer, Actualizar, Eliminar) sobre un modelo específico. Estos controladores incluyen métodos predefinidos que cubren las acciones más comunes al interactuar con los datos, haciendo que el proceso de desarrollar funciones básicas para manejar recursos en una aplicación sea más eficiente y organizado.

Para crear un controlador de recursos, se puede usar el siguiente comando de Artisan:

```
php artisan make:controller RecursoController --resource
```

En los controladores de recursos, como los usados en el dashboard de la aplicación, se define una función CRUD para cada tabla, incluyendo operaciones como crear, modificar, eliminar y visualizar datos. Cada uno de estos métodos está vinculado a vistas correspondientes para la recolección de datos y su posterior almacenamiento en la base de datos.

El uso adecuado de los controladores de recursos facilita la creación de una arquitectura de aplicación coherente y fácil de mantener.

6.4 VISTAS

Las vistas definen la capa de presentación de la aplicación web. Laravel utiliza Blade como motor de plantillas predeterminado, el cual permite integrar lógica sencilla directamente en las vistas sin comprometer la claridad del HTML. A través de componentes, layouts y directivas personalizadas, Blade facilita la construcción de interfaces reutilizables y dinámicas.

Personalizar las vistas de acuerdo con los requerimientos de la aplicación permite ofrecer una experiencia de usuario consistente, accesible y visualmente coherente con la identidad de la plataforma.

6.4.1 Definición

En Laravel, las vistas son archivos Blade (.blade.php) que definen la interfaz de usuario de la aplicación web. Estas vistas permiten combinar HTML con lógica sencilla mediante directivas, facilitando la presentación dinámica de datos.

Los archivos de vista se almacenan en el directorio resources/views.

Estructura de una vista básica de Blade :

```
<!-- resources/views/mi_vista.blade.php -->

<!DOCTYPE html>
<html>
<head>
    <title>Mi Vista</title>
</head>
<body>
    <h1>Hola, {{ $nombre }}</h1>
</body>
</html>
```

Este archivo muestra un saludo personalizado utilizando una variable pasada desde el controlador.

6.4.2 Blade Templanting Engine

Laravel utiliza Blade como motor de plantillas predeterminado para la construcción de vistas. Blade permite incrustar variables y estructuras de control directamente en archivos HTML mediante una sintaxis clara y concisa. Además, proporciona soporte

para la herencia de plantillas, componentes reutilizables y otras funcionalidades que simplifican la organización del código de presentación.

Estructura básica de Blade:

```
<h2>{{ $titulo }}</h2>

@if($mostrar)
    <p>Este es un bloque condicional</p>
@endif
```

Este fragmento muestra cómo Blade permite insertar variables ({{ $titulo }}) y controlar el flujo con directivas como @if.

6.4.3 Compartir datos con vistas

En Laravel, los datos pueden ser enviados desde el controlador hacia una vista utilizando la función view() junto con compact(), with() o pasando un array asociativo. Esto permite que las vistas accedan a la información necesaria para su representación dinámica.

Ejemplo:

```
public function mostrarVista()
{
    $nombre = "Juan";
    return view('mi_vista', compact('nombre'));
}
```

En este ejemplo, la variable $nombre es compartida con la vista mi_vista, donde podrá ser accedida mediante {{ $nombre }}.

6.4.4 Herencia de plantillas

Blade permite la reutilización de estructuras HTML mediante herencia de plantillas. La directiva @extends se utiliza para indicar que una vista hereda de una plantilla base, y @section junto con @yield permiten definir y renderizar secciones específicas del contenido.

Plantilla base: resources/views/layout.blade.php

```
<html>
    <head>
        <title>@yield('titulo')</title>
    </head>
    <body>
        @yield('contenido')
    </body>
</html>
```

Vista que hereda: resources/views/mi_vista.blade.php

```
@extends('layout')

@section('titulo', 'Mi Vista')

@section('contenido')
    <h1>Hola, {{ $nombre }}</h1>
@endsection
```

Este enfoque permite mantener un diseño consistente en toda la aplicación y facilita la mantenibilidad de la interfaz.

6.4.5 Componente y slots

Laravel proporciona componentes de Blade y slots para encapsular y reutilizar fragmentos de interfaz de usuario en múltiples vistas. Esto permite una estructura más limpia, modular y mantenible.

Componente base: resources/views/components/componente.blade.php

```
<div class="box">
    <h2>Componente</h2>
    {{ $slot }}
</div>
```

Uso del componente en una vista: resources/views/mi_vista.blade.php

```
<x-componente>
    <p>Contenido del componente</p>
</x-componente>
```

El contenido definido entre las etiquetas <x-componente> se inserta automáticamente en la variable especial $slot, lo que permite personalizar el interior del componente sin duplicar su estructura.

6.4.6 Directivas de control de flujos

Blade incluye directivas para manejar estructuras de control de flujo, como @if, @foreach y @while. Estas permiten integrar lógica condicional y de repetición dentro de las vistas de forma clara y legible, manteniendo la separación entre la lógica de aplicación y la presentación.

@if:

```
@foreach($usuarios as $usuario)
    <p>{{ $usuario->nombre }}</p>
@endforeach
```

@foreach:

```
@if($activo)
    <p>Este bloque se muestra si la condición es verdadera.</p>
@else
    <p>Este bloque se muestra si la condición es falsa.</p>
@endif
```

@while:

```
@php $i = 0; @endphp
@while($i < 3)
    <p>Iteración {{ $i }}</p>
    @php $i++; @endphp
@endwhile
```

6.4.7 Escapado de datos

Laravel escapa automáticamente las salidas de datos en las vistas Blade para prevenir ataques de tipo Cross-Site Scripting (XSS). Esto asegura que cualquier contenido dinámico mostrado en la interfaz sea seguro por defecto.

```
<p>{{ $htmlEscapado }}</p> <!-- El contenido es escapado -->
<p>{!! $htmlNoEscapado !!}</p> <!-- El contenido no es escapado
-->
```

Cuando es necesario renderizar HTML directamente (por ejemplo, contenido enriquecido proveniente de la base de datos), se puede utilizar la sintaxis {!! !!}, aunque debe hacerse con precaución para evitar vulnerabilidades. Este mecanismo forma parte esencial de la capa de presentación en Laravel, promoviendo tanto seguridad como una experiencia de usuario consistente.

6.5 DEFINICIÓN DE RUTAS

El archivo de rutas en Laravel define cómo se deben manejar las solicitudes HTTP. Organizar y nombrar las rutas de manera lógica facilita la navegación y la comprensión del flujo de la aplicación.

En Laravel 12, las rutas se definen en el archivo routes/web.php para rutas web y en routes/api.php para rutas API. Las rutas pueden asociarse tanto a funciones anónimas como a métodos de controladores.

Ruta en routes/web.php:

```
use App\Http\Controllers\MiControlador;

Route::get('/ruta', [MiControlador::class, 'metodo']);
```

Define una ruta GET para la URL /ruta, que se dirige al método “metodo” del controlador “MiControlador”.

De esta manera, las rutas se pueden estructurar de forma eficiente para que la aplicación web sea fácil de mantener y ampliar.

6.5.1 Parámetros en rutas

Se pueden definir rutas con parámetros para capturar segmentos de la URI. Estos parámetros se pasan automáticamente al controlador o a la función de la ruta, permitiendo una mayor flexibilidad en el manejo de solicitudes.

Ruta con Parámetro:

```
Route::get('/usuario/{id}', function ($id) {
    // Lógica para mostrar el usuario con el ID proporcionado
});
```

En este caso, el valor del parámetro {id} se pasa a la función o controlador como $id, permitiendo realizar operaciones basadas en ese valor, como mostrar un usuario específico.

6.5.2 Nombre de rutas

Asignar nombres a las rutas facilita su referencia en otras partes de la aplicación, como la generación de URLs o redirecciones. Esto permite mantener el código más limpio y manejable.

Ruta con Nombre:

```
Route::get('/dashboard', [MiControlador::class, 'dashboard'])-
>name('dashboard');
```

En este caso, la ruta '/dashboard' está asociada al método dashboard del MiControlador, y se le asigna el nombre 'dashboard'. Este nombre puede utilizarse para generar la URL de la ruta de manera sencilla en otras partes de la aplicación.

6.5.3 Grupos de rutas

Los grupos de rutas permiten aplicar configuraciones compartidas como middleware, prefijos o namespaces a varias rutas, reduciendo la redundancia y mejorando la organización.

```
use App\Http\Controllers\PerfilController;
use App\Http\Controllers\ConfiguracionController;

Route::middleware(['auth'])->group(function () {
    Route::get('/perfil', [PerfilController::class, 'index']);
    Route::get('/configuracion',
[ConfiguracionController::class, 'index']);
});
```

Este enfoque permite definir rutas con requisitos comunes (como autenticación) de forma más limpia y centralizada.

6.5.4 Rutas con middleware

Laravel permite aplicar middleware directamente a rutas individuales, lo que permite ejecutar ciertas acciones antes o después de procesar la solicitud, como verificar la autenticación o roles.

```
use App\Http\Controllers\PerfilController;

Route::get('/perfil', [PerfilController::class, 'index'])-
>middleware('auth');
```

Esta técnica es útil cuando solo se necesita proteger rutas específicas sin agruparlas.

6.5.5 Rutas con vistas

Laravel permite definir rutas que retornan directamente una vista Blade sin necesidad de utilizar un controlador, útil para páginas estáticas o simples.

```
Route::view('/bienvenida', 'bienvenida');
```

Esta opción simplifica la estructura del código cuando no se requiere lógica adicional para renderizar la vista.

6.5.6 Rutas con controladores de recursos

Laravel permite registrar múltiples rutas para operaciones CRUD de manera automática mediante controladores de recursos. Esto simplifica la gestión de modelos como productos, usuarios u órdenes.

```
Route::resource('productos', ProductoController::class);
```

Esta instrucción genera rutas como index, create, store, show, edit, update y destroy, todas asociadas al controlador correspondiente.

6.5.7 Rutas con parámetros opcionales

Laravel permite definir rutas con parámetros opcionales usando el signo de interrogación (?) en la URI. Esto es útil cuando un recurso puede mostrarse con o sin un identificador específico.

```
Route::get('/perfil/{id?}', function ($id = null) {
    // Lógica para mostrar el perfil, el ID es opcional
});
```

Si no se proporciona un ID, se puede definir un comportamiento por defecto dentro del cierre.

6.5.8 Generación de URLs

Laravel permite generar URLs de forma dinámica utilizando el nombre de la ruta o su referencia a controladores, lo que facilita la navegación interna de la aplicación y evita codificar URLs manualmente.

```
$url = route('dashboard');
```

Esto permite acceder a la URL asociada a la ruta nombrada dashboard, mejorando la mantenibilidad y claridad del código.

6.5.9 Rutas con middleware groups

Laravel permite agrupar rutas bajo un conjunto común de middleware, como auth y verified, lo que simplifica la protección de múltiples rutas con los mismos requisitos de acceso.

```
Route::middleware(['auth', 'verified'])->group(function () {
    Route::get('/dashboard', [DashboardController::class,
'index']);
    Route::get('/perfil', [PerfilController::class, 'index']);
});
```

Este enfoque permite una estructura más limpia y coherente en la definición de rutas protegidas. Las rutas cumplen un papel fundamental en la arquitectura del sistema, al ser el punto de entrada a las funcionalidades y reflejar la lógica del negocio; por lo tanto, su diseño debe ser claro, estructurado y mantenible.

6.6 SEGURIDAD EN EL DESARROLLO

Aplicar buenas prácticas de seguridad durante el desarrollo es fundamental para proteger la aplicación contra amenazas comunes. Laravel incorpora mecanismos integrados para validar formularios, prevenir ataques CSRF y gestionar la autenticación de forma segura.

6.6.1 Generación de la llave de aplicación

Laravel emplea una clave única para cifrar datos sensibles, ubicada en el archivo .env bajo el parámetro APP_KEY. Esta clave se genera automáticamente al instalar el framework mediante el comando:

```
php artisan key:generate
```

Es indispensable que esta clave sea única, permanezca secreta y no se comparta entre entornos, ya que garantiza la seguridad en procesos como el cifrado de contraseñas, tokens y otros datos sensibles.

6.6.2 Protección CSRF

Laravel protege contra ataques CSRF (Cross-Site Request Forgery) de forma predeterminada utilizando el middleware VerifyCsrfToken. Este middleware asegura que las solicitudes de tipo POST, PUT, PATCH, y DELETE incluyan un token CSRF válido. La verificación se realiza comparando el token generado por Laravel con el token que se envía en la solicitud, lo que ayuda a prevenir que un atacante realice acciones en nombre del usuario sin su consentimiento.

Un ejemplo básico de uso en formularios sería:

```
<form method="POST" action="/ruta">
    @csrf
    <!-- Otros campos del formulario -->
</form>
```

El uso de @csrf en el formulario genera automáticamente el campo oculto con el token CSRF, protegiendo así la solicitud contra este tipo de ataques.

6.6.3 Middleware de seguridad

Laravel incluye varios middleware de seguridad que permiten proteger las rutas de la aplicación. Algunos de los middleware más comunes son:

- **auth**: Este middleware asegura que solo los usuarios autenticados puedan acceder a ciertas rutas. Si un usuario no está autenticado, será redirigido a la página de inicio de sesión.

- **verified**: Este middleware verifica que el usuario haya confirmado su dirección de correo electrónico antes de poder acceder a ciertas rutas. Esto es útil para garantizar que los usuarios tengan una cuenta verificada antes de acceder a funcionalidades críticas.

Ejemplo de uso de middleware de seguridad:

```
Route::middleware(['auth', 'verified'])->group(function () {
    Route::get('/dashboard', 'DashboardController@index');
    Route::get('/perfil', 'PerfilController@index');
});
```

En este ejemplo, ambas rutas, /dashboard y /perfil, estarán protegidas por los middleware auth y verified. Esto significa que solo los usuarios autenticados y cuya dirección de correo electrónico haya sido verificada podrán acceder a ellas.

6.6.4 protección XSS

Laravel protege contra ataques de tipo XSS (Cross-Site Scripting) utilizando el sistema de plantillas Blade. Cuando se imprime una variable con {{ $variable }}, Laravel escapa automáticamente cualquier contenido HTML o JavaScript potencialmente malicioso.

```
<p>{{ $contenido }}</p>
```

sí $contenido contiene algo como <script>alert('Ataque');</script>, Laravel lo transformará en:

```
<p>&lt;script&gt;alert('Ataque');&lt;/script&gt;</p>
```

Esto evita que el navegador ejecute el script.

Si, por alguna razón, se necesita renderizar HTML de forma segura y controlada, se puede usar {!! $contenido !!} — aunque esto debe usarse con precaución, ya que omite el escape automático.

6.6.5 Configuración de Seguridad en Archivo config/app.php

El archivo config/app.php contiene configuraciones clave que influyen en la seguridad de una aplicación Laravel. Algunas de las más relevantes incluyen:

Debug

```
'debug' => env('APP_DEBUG', false),
```

Debe estar en false en producción para evitar que se muestren errores detallados al usuario. Esto se debe a que tenerlo en true en producción puede exponer información sensible.

URL

```
'url' => env('APP_URL', 'http://localhost'),
```

Define la URL base de la aplicación, esta debe estar correctamente configurada para evitar problemas con redirecciones, rutas o generación de enlaces inseguros.

Recomendaciones

En producción, definir en el archivo .env:

```
APP_DEBUG=false
APP_URL=https://tudominio.com
```

Validar que otros entornos (testing, staging) tengan configuraciones apropiadas para su propósito.

6.6.6 Contraseñas seguras

Laravel asegura el almacenamiento de contraseñas utilizando el algoritmo bcrypt, que es resistente a ataques de fuerza bruta debido a su naturaleza de procesamiento intensivo.

Hash de contraseñas

No se recomienda hacer hashing directamente así:

```
'password' => bcrypt('contraseña segura'),
```

En su lugar, Laravel proporciona la facade Hash para una implementación más clara y flexible:

```
use Illuminate\Support\Facades\Hash;

$user->password = Hash::make('contraseña segura');
```

Verificación de contraseñas

Para verificar su contraseña:

```
if (Hash::check('entrada_usuario', $user->password)) {
    // La contraseña es válida
}
```

Restablecimiento seguro de contraseñas

Laravel ofrece:

- Generación de tokens únicos y temporales.
- Enlaces enviados por correo.
- Middleware que protege las rutas de restablecimiento.

Recomendación

No almacenar nunca contraseñas sin cifrar. Laravel ya lo hace correctamente por defecto al registrar usuarios mediante Auth::routes() o Jetstream/Fortify.

6.6.7 Políticas de contraseña

Laravel permite definir reglas de validación para contraseñas en formularios, normalmente dentro de controladores o archivos FormRequest. Estas políticas garantizan contraseñas más seguras al momento del registro o cambio.

```
$request->validate([
    'password' => [
        'required',
        'string',
        'min:8',              // Longitud mínima
        'confirmed',          // Confirmación del campo
password_confirmation
        Password::min(8)
            ->mixedCase()     // Mayúsculas y minúsculas
            ->letters()       // Requiere letras
            ->numbers()       // Requiere números
            ->symbols()       // Requiere símbolos
            ->uncompromised(),// Verifica que no esté en listas
de contraseñas filtradas
    ],
]);
```

Requiere use Illuminate\Validation\Rules\Password;

Configuracion con reglas personalizadas

También es posible crear una regla personalizada (como password_strength) con php artisan make:rule PasswordStrength, y luego implementarla en la validación.

6.6.8 Logs y auditoria

Laravel proporciona herramientas integradas para registrar eventos del sistema y facilitar auditorías de seguridad, utilizando Log y paquetes como spatie/laravel-activitylog.

Aplicación básica con el facade Log:

```
use Illuminate\Support\Facades\Log;

Log::info('Usuario accedió al sistema', [
    'user_id' => auth()->id(),
    'ip' => request()->ip(),
    'time' => now(),
]);
```

Esto genera una entrada en el archivo storage/logs/laravel.log, útil para rastrear accesos, errores u operaciones críticas.

Auditoria con Spatie Activitylog

Para realizar una auditoria avanzada:

Instalar

```
composer require spatie/laravel-activitylog
```

Publicar config

```
php artisan vendor:publish --
provider="Spatie\Activitylog\ActivitylogServiceProvider"
```

Registrar actividades

```
activity()
    ->causedBy(auth()->user())
    ->log('El usuario accedió al sistema');
```

Esto almacena auditoría en la tabla activity_log, con usuario, acción y contexto.

6.6.9 Actualizaciones de dependencias

Mantener actualizadas las dependencias, como el framework Laravel y sus paquetes asociados, es esencial para aplicar parches de seguridad y mejoras funcionales.

Este comando:

```
composer update
```

Realiza las siguientes acciones:

- Revisa el archivo composer.json, que contiene las dependencias declaradas y sus versiones permitidas.
- Consulta el archivo composer.lock, que guarda las versiones instaladas actualmente.
- Descarga las versiones más recientes permitidas por las restricciones de versión definidas en composer.json.
- Actualiza el archivo composer.lock con las nuevas versiones instaladas
- Aplica cambios automáticamente al directorio vendor/, donde residen todas las dependencias del proyecto.

Recomendaciones

- Ejecutar este comando en un entorno local o de pruebas antes de aplicarlo en producción.
- En servidores VPS (Virtual Private Server), se puede realizar directamente si se cuenta con acceso y control del entorno.

- Siempre realizar respaldos y pruebas de compatibilidad antes de actualizar en un entorno en vivo.

6.6.10 Seguridad en base de datos

Laravel incorpora mecanismos que previenen ataques comunes como la inyección SQL, facilitando una interacción segura con la base de datos mediante su ORM Eloquent o mediante consultas preparadas. Ejemplo con consultas preparadas:

```
$usuarios = DB::select("SELECT * FROM usuarios WHERE id = :id",
['id' => $id]);
```

Alternativa con Eloquent:

```
$usuario = Usuario::find($id);
```

Ambos métodos son seguros, pero se recomienda usar Eloquent siempre que sea posible por su legibilidad y protección adicional al manejar modelos.

6.6.11 Configuración CORS

CORS (Cross-Origin Resource Sharing) es un mecanismo que permite controlar cómo los recursos en un servidor pueden ser solicitados desde otro dominio. Laravel facilita la configuración de CORS para proteger las rutas de la aplicación al permitir o restringir solicitudes de otros dominios.

```
return [
    'allowed_origins' => ['http://mi_otro_dominio.com'],
    'allowed_methods' => ['GET', 'POST', 'PUT', 'DELETE'],
    'allowed_headers' => ['Content-Type', 'Authorization'],
];
```

Una vez configurado config/cors.php, Laravel manejará automáticamente las solicitudes CORS. En caso de ser necesario, también puedes agregar middleware CORS adicional para personalizar más aún el comportamiento.

7 PRUEBAS

Las pruebas son fundamentales en el desarrollo de aplicaciones para garantizar que el código funcione correctamente y que los cambios no rompan funcionalidades existentes.

7.1 TIPO DE PRUEBAS

Laravel 12 permite realizar pruebas unitarias (para funciones o métodos individuales), pruebas de integración (para verificar la interacción entre componentes) y pruebas de características (que simulan el uso real del sistema desde el navegador o API).

7.2 PRUEBAS UNITARIAS

Las pruebas unitarias en Laravel 12 validan funciones o métodos específicos de forma aislada. Se implementan con PHPUnit, lo que permite escribir y ejecutar pruebas fácilmente dentro del entorno del framework. Para correr las pruebas, solo es necesario ejecutar el comando:

```
php artisan test
```

7.3 PRUEBAS DE INTEGRACIÓN

Las pruebas de integración en Laravel 12 verifican cómo interactúan varios componentes entre sí. Se pueden simular solicitudes HTTP con métodos como get(), post() y verificar respuestas, lo que permite evaluar el comportamiento completo del sistema.

7.4 PRUEBAS DE CARACTERÍSTICAS (FEATURE TESTS)

Las pruebas de características permiten verificar funcionalidades completas, como el flujo de autenticación o envío de formularios. Laravel proporciona métodos para simular interacciones del usuario y validar resultados esperados en la interfaz o respuesta HTTP.

7.5 FACTORY PARA DATOS DE PRUEBA

Laravel permite generar datos de prueba con model factories, facilitando la creación de registros ficticios en la base de datos durante las pruebas. Esto se realiza comúnmente con el paquete Faker incluido.

```
// Ejemplo básico en una factory
public function definition(): array
{
    return [
        'nombre' => $this->faker->name(),
        'email' => $this->faker->unique()->safeEmail(),
        // ... otras columnas
    ];
}
```

Estas factories pueden combinarse con seeder o pruebas para poblar tablas y simular escenarios reales de forma rápida.

7.6 ASSERTIONS Y MÉTODOS DE EVALUACIÓN

Laravel ofrece métodos de aserción que permiten validar el comportamiento de la aplicación durante las pruebas. Algunos ejemplos comunes son assertStatus, assertSee y assertDatabaseHas, los cuales verifican el estado HTTP, la presencia de texto en la respuesta y registros en la base de datos, respectivamente.

```
// Ejemplo de aserción
public function test_usuario_puede_ver_pagina()
{
    $response = $this->get('/ruta');
    $response->assertStatus(200);
    $response->assertSee('Texto esperado');
}
```

Estas herramientas son clave para confirmar que la aplicación responde correctamente ante diferentes escenarios.

7.7 MIGRACIONES DE PRUEBAS

Laravel permite usar migraciones específicas durante las pruebas para asegurar un entorno de base de datos limpio y consistente. Al ejecutar pruebas, Laravel utiliza una base de datos en memoria (como SQLite) o una base temporal, y ejecuta automáticamente las migraciones antes de iniciar cada prueba si se usa el trait RefreshDatabase.

7.8 GRUPOS Y ETIQUETAS DE PRUEBAS

Laravel permite organizar pruebas utilizando anotaciones como @group para etiquetar pruebas o conjuntos de pruebas, lo que facilita su ejecución selectiva. Por ejemplo, se puede ejecutar solo un grupo específico con php artisan test --group=nombre_grupo.

7.9 PRUEBAS CONTINUAS (CONTINUOS TESTING)

Laravel permite integrar pruebas continuas (Continuous Testing) mediante herramientas como GitHub Actions o GitLab CI. Estas herramientas ejecutan automáticamente las pruebas cada vez que se realiza un cambio en el código, lo que ayuda a identificar errores tempranamente y asegura que las nuevas funcionalidades no afecten el rendimiento o la estabilidad de la aplicación.

Al configurar un archivo de CI, como un .yml en GitHub Actions, las pruebas se ejecutan en cada push o pull request, lo que optimiza el flujo de trabajo y garantiza que el código se mantenga libre de errores a lo largo del desarrollo.

8 MANTENIMIENTO

El mantenimiento de una aplicación Laravel implica mantener actualizado tanto el framework como sus dependencias. Las actualizaciones frecuentes son esenciales para garantizar la seguridad y el buen funcionamiento de la aplicación.

8.1 ACTUALIZACIONES DEL FRAMEWORK

Para actualizar Laravel y sus paquetes, se utilizan los siguientes comandos de Artisan:

```
composer update
php artisan update
```

Es recomendable realizar estas actualizaciones primero en un entorno de desarrollo antes de desplegar los cambios en producción para evitar interrupciones.

8.2 GESTIÓN DE DEPENDENCIAS

La gestión de dependencias en Laravel se realiza mediante el archivo composer.json, que define las bibliotecas y versiones necesarias para el funcionamiento de la aplicación. Para mantener un entorno estable, es importante:

- Revisar regularmente las dependencias para asegurar que estén actualizadas y sean compatibles entre sí.
- Evitar versiones inestables de paquetes o librerías, optando por versiones estables que hayan sido probadas.
- Comprobar la compatibilidad de las nuevas versiones de dependencias con la versión de Laravel y otros paquetes utilizados.
- Usar el comando composer update para actualizar las dependencias, y siempre probar los cambios en un entorno de desarrollo antes de aplicarlos en producción.

Al seguir estas prácticas, se facilita la gestión de dependencias y se minimizan los riesgos de incompatibilidades o vulnerabilidades.

8.3 COPIAS DE SEGURIDAD (BACKUPS)

Es esencial establecer copias de seguridad periódicas. Laravel ofrece Laravel Backup para realizar respaldos automáticos de bases de datos y archivos. Se recomienda almacenar las copias en la nube o servidores externos y verificar que puedan restaurarse correctamente en caso de emergencia.

8.4 MONITOREO DE RENDIMIENTO

Implementar herramientas como Laravel Telescope permite monitorear el rendimiento de la aplicación, identificar cuellos de botella y optimizar áreas clave. Esto ayuda a mantener una experiencia de usuario fluida y un rendimiento eficiente.

8.5 ACTUALIZACIÓN DE DEPENDENCIAS DE SEGURIDAD

Es importante revisar y actualizar las dependencias de seguridad de manera regular. Laravel y herramientas como la base de datos de asesoría de GitHub ofrecen alertas sobre vulnerabilidades conocidas. Para actualizar las dependencias de seguridad, se utiliza el comando composer update y se verifica el archivo composer.json para asegurarse de que no haya versiones vulnerables en uso.

8.6 GESTIÓN DE ERRORES Y REGISTRO

Configurar un sistema de registro de errores efectivo es esencial para detectar y solucionar problemas rápidamente. Laravel usa Monolog para manejar los registros de errores, permitiendo personalizar los canales de log y almacenar los errores de forma organizada. Se debe asegurarse de que el archivo .env tenga configurado el nivel adecuado de logs, como LOG_LEVEL=error, y configurar correctamente los canales de registro en config/logging.php.

8.7 OPTIMIZACIÓN DE LA BASE DE DATOS

La optimización de consultas es crucial para mantener un buen rendimiento. Laravel Debugbar permite analizar consultas SQL y detectar posibles cuellos de botella. Además, se recomienda usar Eloquent de manera eficiente, evitando consultas innecesarias, utilizando eager loading (with()) para prevenir el N+1 query problem y creando índices en las columnas más consultadas para mejorar la velocidad de acceso a datos.

8.8 PRUEBAS DE REGRESIÓN

Las pruebas de regresión aseguran que las nuevas funcionalidades no rompan las existentes. Laravel permite automatizar estas pruebas utilizando PHPUnit, lo que facilita la verificación continua de la integridad del sistema. Al agregar nuevas funcionalidades o realizar cambios, ejecutar estas pruebas ayuda a identificar cualquier impacto no deseado en el comportamiento de la aplicación.

8.9 GESTIÓN DE VERSIONES

Usar un sistema de control de versiones como Git es fundamental para gestionar el desarrollo de una aplicación. Permite registrar cambios, colaborar de manera eficiente y revertir a versiones anteriores si surgen problemas. Seguir buenas prácticas de Git, como la creación de ramas para nuevas características y la escritura de mensajes de commit claros, facilita un mantenimiento organizado y controlado.

8.10 DOCUMENTACIÓN ACTUALIZADA

Mantener la documentación actualizada es esencial para facilitar futuros desarrollos y permitir que nuevos miembros del equipo comprendan la arquitectura de la aplicación. Herramientas como Swagger son útiles para documentar APIs en Laravel.

9 LICENCIAS DE USO DE HERRAMIENTAS Y COMPONENTES

El sistema MOVEBREAK ha sido desarrollado utilizando herramientas y librerías de código abierto, tales como Laravel, Jetstream, Livewire, TensorFlow, entre otras. Estas tecnologías se encuentran bajo licencias como MIT, Apache 2.0 y GPL v2, que establecen condiciones específicas para su uso, modificación y redistribución. Actualmente, el software es de uso interno exclusivo, por lo cual no existe ninguna obligación legal de liberar su código fuente, incluso si se han utilizado componentes con licencia GPL. Sin embargo, si en el futuro se decide redistribuir, comercializar o entregar este software a terceros, es importante considerar que:

- Las librerías con licencia GPL (como DomPDF o MySQL) obligan a que el código fuente completo derivado sea también puesto a disposición del receptor, si el software se redistribuye.
- Las licencias MIT y Apache 2.0, utilizadas por la mayoría del stack del sistema, permiten la distribución sin obligación de liberar el código, pero requieren mantener los avisos de atribución.

Se recomienda, ante cualquier intención de distribución externa, realizar una revisión de dependencias y considerar reemplazar librerías con licencias restrictivas por otras más permisivas si se desea mantener el código cerrado.

Licencias oficiales:

- MIT License: https://opensource.org/licenses/MIT
- Apache License 2.0: https://www.apache.org/licenses/LICENSE-2.0
- GNU GPL v2: https://www.gnu.org/licenses/old-licenses/gpl-2.0.html

10 REFERENCIAS

A continuación, se listan las referencias utilizadas durante el desarrollo de este documento y que han guiado el proceso, ya sea por la inclusión de nuevas funcionalidades o por dificultades encontradas en el desarrollo de estas.

- Laravel. (2025). Laravel Documentation. https://laravel.com/docs/12.x
- MariaDB Foundation. (2025). MariaDB Knowledge Base. https://mariadb.com/kb/en/
- Apache. (2025). Apache HTTP Server Documentation. https://httpd.apache.org/docs/
- MySQL. (2025). MySQL Documentation. https://dev.mysql.com/doc/
- W3Schools. (2025). PHP Tutorial. https://www.w3schools.com/php/
- PHPUnit. (2025). PHPUnit Documentation. https://phpunit.de/documentation.html
- Laravel Dusk. (2025). Laravel Dusk Documentation. https://laravel.com/docs/12.x/dusk
- SeleniumHQ. (2025). Selenium Documentation. https://www.selenium.dev/documentation/en/
- OWASP. (2025). Open Web Application Security Project. https://owasp.org/
- Stack Overflow. (2025). Online Developer Community. https://stackoverflow.com/