República Bolivariana De Venezuela
Ministerio del poder popular para la educación
Universidad Nacional Experimental
De los Llanos Occidentales
“Ezequiel Zamora”.
Barinas Edo Barinas
Vicerrectorado de Planificación y Desarrollo Social.
Programa de Ciencias Básicas y Aplicadas.
Subprograma Ingeniera Informática.

Asociación de Karate DO DE Barinas (Asokarate)

Docente: Franklin España

Bachilleres:

- Calleja Williams
  CI: 30.823.106
- Herrera Jennyfer
  CI: 30.194.444
- Hernández Ashly
  CI: 31.439.830
- Montilva Febe
  CI: 30.609.455
- Rivas Jose Gabriel
  CI: 29.639.466
- Rodríguez Alexander
  CI: 30.409.477

Contenido

Introducción

El Karate es un arte marcial que exige disciplina y técnica, un alto nivel de preparación física y mental. Los eventos de Karate, ya sean competencias de combate o coreografía de combate, simbolizan momentos esenciales a la hora de evaluación y crecimiento de los practicantes. Para la organización eficiente de estos eventos incluye el registro de participantes, árbitros, resultados y un manejo eficiente de las diferentes categorías y tatamis es crucial para avalar el orden y la evolución de las competencias.

En este aspecto, la aplicación de registro de eventos de Karate fue creada para cubrir las necesidades de estrategias y operativas asociadas para la realización de un torneo o campeonato. No obstante Con esta plataforma, los organizadores pueden registrar de una manera fácil los datos de los competidores, y así poder asignar de una manera correcta a cada participante a sus categorías correspondientes. De igual forma, permite el registro detallado de los tatamis y el lugar correspondiente donde se desarrollan las competencias, y la administración de árbitros que supervisan y califican los combates.

Asimismo, la aplicación incluye funcionalidades avanzadas como la visualización de resultados en tiempo real, lo que facilita el seguimiento continuo del evento por parte de participantes, público y organizadores, similar a otros deportes profesionales. La integración de sistemas para generar combates automáticos y una tabla de clasificación mejora la fluidez y eficiencia del evento, reduciendo posibles errores manuales y agilizando la toma de decisiones.

Es por esto que para el desarrollo de esta aplicación se emplean tecnología moderna y robusta como HTML, CSS, PHP con LARAVEL, JavaScript y MySQL en la gestión de bases de datos. Estas tecnologías permiten la realización de una plataforma accesible y eficiente, intuitiva y escalable, que es apta a las dinámicas específicas de los eventos de karate.

Definición de Asociación de Karate DO DE Barinas (Asokarate)

La aplicación de registro de eventos de Karate fue creada para cubrir las necesidades de estrategias y operativas asociadas para la realización de un torneo o campeonato. No obstante Con esta plataforma, los organizadores pueden registrar de una manera fácil los datos de los competidores, y así poder asignar de una manera correcta a cada participante a sus categorías correspondientes. De igual forma, permite el registro detallado de los tatamis y el lugar correspondiente donde se desarrollan las competencias, y la administración de árbitros que supervisan y califican los combates.

Especificaciones de Requerimientos de Asociación de Karate DO DE Barinas (Asokarate)

1. Registro de Eventos:

- Kumite (Combate): Registro de participantes con información relevante como edad, peso y cinta.
- Kata (Coreografía de Combate): Registro de participantes y sus respectivas coreografías.
- Tatami (Lona de Combate): Registro de los tatamis utilizados en las competencias.

2. Categorías de Competencia:

La aplicación permitirá la búsqueda de eventos por tatami, facilitando el acceso a la información de:

- Participantes
- Árbitros
- Puntos
- Registro de Ganadores

3. Resultados en Tiempo Real:

- Visualización de resultados en tiempo real, similar a los sistemas utilizados en eventos deportivos como el fútbol.
- Generación automática de combates y tabla de clasificación del evento.

4. Gestión de Combates:

Registro de combate con tiempos específicos según la categoría:

- Kumite: 3 minutos
- Kata: 5 minutos

- Finalización de combates: se determina un ganador cuando hay una diferencia de 8 puntos o al finalizar el tiempo.
- Sistema de puntuación y descalificaciones:
  - Senshu: el competidor que marca el primer punto.
  - Descalificaciones por acumulación de faltas (5 faltas resultan en descalificación).

5. Interfaz de Usuario:

- Súper Admin: Panel de control con acceso a las tablas de registro de dojos y eventos, así como la opción de crear nuevos eventos.
- Árbitros: Acceso a los eventos en los que están asignados, con una vista de estadísticas de los jugadores y un tablero para calificar.

Metodología Empleada para el Desarrollo del Software

Una metodología de desarrollo de software se define como el conjunto de principios, técnicas, herramientas y procesos utilizados para estructurar, planificar y controlar el proceso de creación de un sistema de software. Dada la naturaleza secuencial y bien definida de las etapas requeridas para la implementación de las funcionalidades clave de la plataforma (desde la planificación inicial hasta la implementación y pruebas).

Existen varias metodologías de desarrollo de software, cada una con sus propias características y enfoques. Cada metodología tiene su propio conjunto de prácticas y herramientas que pueden adaptarse según las necesidades específicas del proyecto y del equipo de desarrollo. La elección de una metodología adecuada depende de factores como el tamaño del proyecto, la naturaleza del software, los requisitos del cliente y la experiencia del equipo.

Para el desarrollo del Software **(Nombre),** hemos implementado la siguiente metodología cascada, qué es un enfoque de gestión de proyectos que se caracteriza por su naturaleza secuencial y lineal. Este modelo se basa en un proceso estructurado en el cual cada fase del proyecto debe completarse en su totalidad antes de avanzar a la siguiente.

Fases del desarrollo

1. **Recopilación de requisitos:**

En esta fase, los requisitos del proyecto se identifican y documentan, el objetivo de esta fase es establecer una base sólida para el proyecto definiendo lo que se debe lograr.

● Se investigó la documentación necesaria para tener las bases reglamentarias que se van a implementar dentro de las funciones de los eventos, así tenemos en cuenta los requisitos necesarios para que la página sea de apoyo real en las competiciones de karate.

● Se recopilaron formatos recientes en la toma de puntuaciones, organización de combates y estadísticas, gracias a esto se tiene una guía y una idea para realizar el diseño de la página fiel a los estándares de las competiciones profesionales.

2. **Diseño:**

Una vez reunidos los requisitos, comienza la fase de diseño. Incluye definir la estructura, los componentes y las experiencias de usuario. En la etapa de diseño, se adoptó la arquitectura Modelo-Vista-Controlador (MVC) para estructurar el sistema en tres capas interconectadas: el Modelo (gestión de datos y lógica de negocio), la Vista (presentación de la interfaz de usuario) y el Controlador (manejo de la interacción y flujo de la aplicación). Esta arquitectura facilita la organización del código, la separación de responsabilidades y, por ende, el desarrollo, la prueba y el mantenimiento. Para visualizar y detallar la estructura y el comportamiento del sistema diseñado bajo este patrón MVC, se emplearán los Diagramas de Modelado Unificado (UML).

UML ofrece un conjunto de notaciones gráficas estándar que permiten representar tanto la estructura estática de los componentes (como las relaciones entre los modelos de datos: torneos, participantes, combates) como la dinámica de las interacciones entre las Vistas, los Controladores y los Modelos durante la ejecución de las funcionalidades.

3. **Implementación:**

En esta fase, el equipo del proyecto ha completado el trabajo de desarrollo real, fabricando los entregables de acuerdo a las especificaciones de diseño previamente establecidas. Durante esta etapa, los desarrolladores han seguido los planes de diseño anteriormente elaborados, traduciendo esos diseños a un código funcional y eficiente.

Se ha estado utilizando la arquitectura Modelo-Vista-Controlador (MVC) para estructurar la aplicación, lo que permitió una separación clara de las responsabilidades y la facilitación del mantenimiento del código. Además se estuvo gestionando el código con ramas de Git, lo que permitió un trabajo colaborativo y muy eficiente, en donde el diseño previamente elaborado se está traduciendo en código funcional. El equipo escribe el código fuente de la aplicación basándose en las

especificaciones detalladas en la etapa de diseño (modelos de datos, interfaces de usuario, lógica de control, estructura de la base de datos), el uso de Git y ramas facilita la gestión de las diferentes funcionalidades en el desarrollo, permitiendo que los miembros del equipo trabajen simultáneamente en distinta características sin interferencias.

Actividades Principales:

- **Configuración del Entorno de Desarrollo:**

Cada desarrollador configura su entorno local con las herramientas necesarias.

- **Creación de la Estructura del Proyecto MVC:**

Se establece la estructura de carpetas y archivos. Esto típicamente incluye directorios para los Modelos, las Vistas y los Controladores.

- **Implementación del Modelo**:
- Se definen las clases que representan las entidades del dominio (Torneo, Participante, Combate, Sorteo, Arbitraje, Categoría, Inscripción).
- Se implementa la lógica para interactuar con la base de datos (ORM o consultas SQL directas) para crear, leer, actualizar y eliminar datos de estas entidades.

- **Desarrollo por Ramas de Git**:

Se utiliza un flujo de trabajo con ramas en Git para gestionar el desarrollo de las diferentes funcionalidades de manera aislada. Cada nueva funcionalidad o corrección de errores se desarrolla en una rama separada.

4. Pruebas

Después de la fase de implementación, se llevaron a cabo diversas pruebas rigurosas para garantizar la calidad y funcionalidad del sistema. Las pruebas incluyeron:

- **Pruebas de Integración:** Estas pruebas verificamos la correcta interacción entre los diferentes módulos y componentes del sistema, asegurando que cada parte funcionara de manera correcta y sin errores.
- **Pruebas del Sistema:** Se evaluó que el sistema cumpliera con todos los requisitos funcionales y no funcionales especificados en la fase de diseño. De manera similar la verificación de la interfaz de usuario, la gestión de datos.
- **Pruebas de Rendimiento:** Se analizó la capacidad del sistema para funcionar correctamente bajo diferentes cargas de trabajo y condiciones de uso.

- **Pruebas de Usabilidad**: En esta prueba nos centramos en evaluar la facilidad con la que los usuarios pueden interactuar con la plataforma y completar sus tareas.

5. Despliegue

La fase de despliegue marcó el momento en que la aplicación estuvo lista para ser lanzada y utilizada en el entorno real. Esta fase se llevó a cabo después de completar con éxito la fase de prueba

6. Mantenimiento

Durante la fase de mantenimiento, el equipo del proyecto proporcionó soporte continuo para abordar cualquier problema que pudiera surgir. El objetivo principal de esta fase fue asegurar que el proyecto continuara funcionando adecuadamente y cumpliendo con las expectativas de los usuarios.

Actividades Realizadas:

- Resolución de errores o problemas identificados en el proyecto.
- Implementación de cambios o mejoras necesarias basadas en los comentarios de los usuarios y nuevos requisitos.

Especificaciones de Requerimientos

Objetivos

- Desarrollar un sistema digital que permita el registro eficiente de participantes, árbitros y eventos de Karate, incluyendo categorías y tatamis.
- Implementar funcionalidades para la gestión de combates de Kumite y coreografías de Kata, considerando tiempos, puntajes y reglas específicas de cada categoría.
- Facilitar la visualización de resultados en tiempo real durante las competencias para mejorar la experiencia de los usuarios y la transparencia del evento.
- Crear una interfaz intuitiva y segura para distintos perfiles de usuarios, incluyendo súper administradores y árbitros, que permita la administración y calificación fácil de los eventos.
- Utilizar tecnologías robustas y escalables como Laravel y MySQL para garantizar la integridad y disponibilidad de los datos almacenados.
- Asegurar la compatibilidad de la aplicación para su uso en dispositivos móviles y escritorio, optimizando la usabilidad en distintas plataformas.

- Implementar mecanismos para el registro y control de faltas, descalificaciones y puntuaciones, siguiendo las normativas oficiales del Karate.
- Permitir la generación automática de combates y la creación dinámica de tablas de clasificación basadas en los resultados registrados.

Alcances

- La aplicación cubrirá la gestión completa de eventos de Karate, desde la creación y registro de eventos hasta la finalización de competencias y registro de ganadores.
- Se registrarán datos detallados de los participantes, árbitros, tatamis y categorías específicas de competición.
- La aplicación ofrecerá funcionalidades para la puntuación en tiempo real y visualización inmediata de resultados, optimizando la experiencia durante las competencias.
- Soportará las reglas oficiales de Karate en cuanto a duración de combates, criterios de finalización y manejo de faltas y descalificaciones.
- La plataforma tendrá perfiles diferenciados para súper administradores y árbitros con privilegios acorde a sus responsabilidades.
- La aplicación se desarrollará como un sistema web responsivo, accesible desde navegadores de escritorio y móviles.
- No se incluye en el alcance el desarrollo de componentes de inteligencia artificial o aprendizaje automático para la evaluación automática de combates.

Limitaciones

- La aplicación depende de la correcta configuración y disponibilidad del entorno web y base de datos, lo que puede afectar su funcionamiento en caso de fallos en infraestructura.
- La evaluación y calificación de combates dependerán de la intervención humana de los árbitros; no se automatizará la valoración técnica de los puntos ni de los movimientos.
- La implementación opcional de tecnologías como Vue.js puede no estar presente en todas las versiones, afectando algunas funcionalidades interactivas.
- El sistema no contempla integración con otros sistemas externos o plataformas de gestión deportiva.
- La aplicación no contempla funciones avanzadas para la gestión financiera o inscripción en línea fuera del sistema interno de eventos.

La seguridad avanzada como autenticación multifactor o cifrado integral no forma parte del desarrollo inicial y debería ser considerado en futuras actualizaciones

Especificaciones de Procedimientos y Herramientas

- IDE: Visual Studio 2025 Community Edition.
- Lenguaje de Backend: Laravel (Php + Blade)
- Lenguaje de Frontend: JavaScript
- Framework: CSS

Ficha Técnica

Características del producto	
Nombre del Producto	Asociación de Karate DO DE Barinas (Asokarate)
Versiones anteriores	N.A
Versión actual	1.0
Descripción del producto	
Descripción general del producto:	Es un sistema de gestión de Eventos para la asociación de Karate DO DE Barinas

	(Asokarate)
Objetivo del producto:	Desarrollar un sistema web para la gestión y creación de Eventos de la asociación de Karate do de Barinas
Arquitectura	
Descripción:	Aplicación cliente-servidor para registro y gestión de eventos de Karate (Kumite y Kata). Frontend con HTML, CSS y JS (Vue.js opcional), Backend en PHP Laravel y base de datos MySQL. Permite registrar participantes, árbitros, tatamis, generar combates, controlar puntos y faltas según reglas oficiales. Visualización en tiempo real de resultados y clasificaciones. Seguridad con roles definidos y acceso restringido. Arquitectura escalable, mantenible y optimizada para dispositivos móviles y escritorio.
Requerimientos del producto	
Requerimientos del sistema (a nivel de cómputo)	
Hardware:	Ordenador Intel i5 , 8Gb de RAM
Software:	Versión Windows 10
Requerimientos del sistema (cliente)	
Hardware:	Ordenador Intel i3 , 2Gb de RAM
Software:	Versión Windows 8
Requerimientos	
Requerimientos funcionales:	• Registro de Participantes • Gestión de Tatamis • Gestión de Árbitros • Generación y Registro de Combates • Visualización en Tiempo Real • Gestión de Eventos • Autenticación y Roles
Requerimientos no funcionales:	• Usabilidad • Rendimiento • Seguridad • Escalabilidad

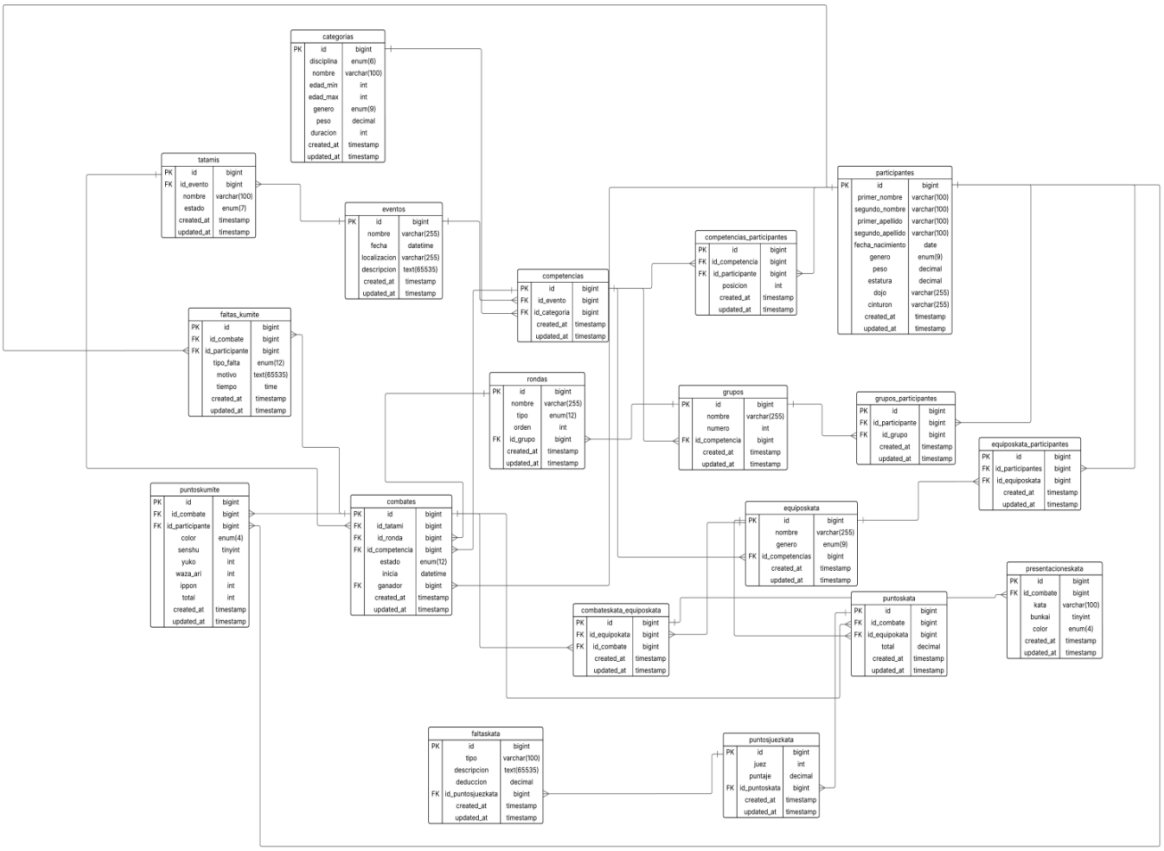
	• Disponibilidad • Mantenibilidad
Clientes del producto:	• Organizadores de Eventos • Árbitros • Participantes • Dojos y Clubes de Karate • Espectadores y Público General • Federaciones y Asociaciones de Karat

Arquitectura del Sistema

Mapa de Navegación

Diseño del Modelo de Datos

Diagrama Entidad-Relación



Descripción de Entidades y Relaciones

Entidades y Atributos

1. **Categorías**

- PK: id

Atributos:

- Disciplina (enum)
- Nombre (varchar(300))
- Estado (int)
- Edad (int)
- Genero (enum)
- Decimal (int)

- creado_at (timestamp)
- actualizado_at (timestamp)

2. **Tatamis**
- PK: id

Atributos:

- id_evento (bigint)
- Estado (enum)
- Creado_at (timestamp)
- Actualizado_at (timestamp)

3. Eventos
- PK: id

Atributos:

- Nombre (varchar(255))
- Fecha (datetime)
- Localización (varchar(255))
- Descripción (text)
- Creado_at (timestamp)
- Actualizado_at (timestamp)

4. **Competencias**
- PK: id

Atributos:

- id_evento (bigint)
- id_categoria (bigint)
- Creado_at (timestamp)
- Actualizado_at (timestamp)

5. Participantes
- PK: id

Atributos:

- primer_nombre (varchar(100))

- segundo_nombre (varchar(100))
- primer_apellido (varchar(100))
- segundo_apellido (varchar(100))
- fecha_nacimiento (date)
- género (enum)
- peso (decimal)
- estatura (decimal)
- dojo (varchar(255))
- cínema (varchar(255))
- creado_at (timestamp)
- actualizado_at (timestamp)

6. **Combates**

- PK: id

Atributos:

- id_tatami (bigint)
- id_competencia (bigint)
- estado (enum)
- id_ron (bigint)
- ganador (bigint)
- creado_at (timestamp)
- actualizado_at (timestamp)

7. **puntos_kumite**

- PK: id

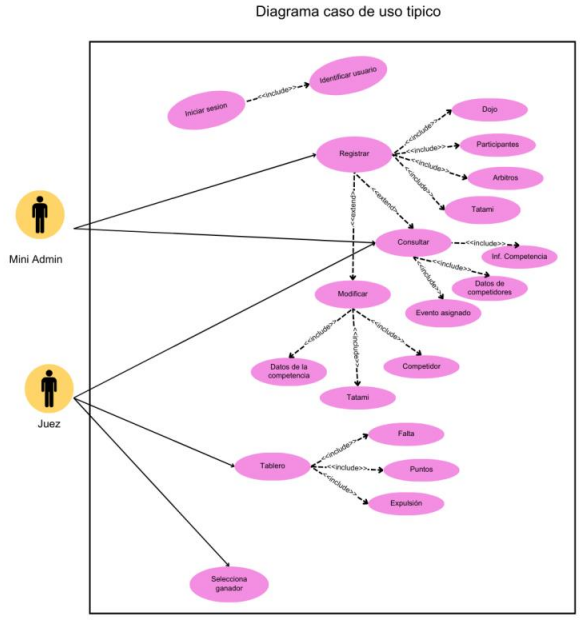
Atributos:

- id_combate (bigint)
- id_participante (bigint)
- color (enum)
- sentido (tinyint)
- yūko (int)
- waza_ari (int)
- ippon (int)
- total (int)
- creado_at (timestamp)

- actualizado_at (timestamp)

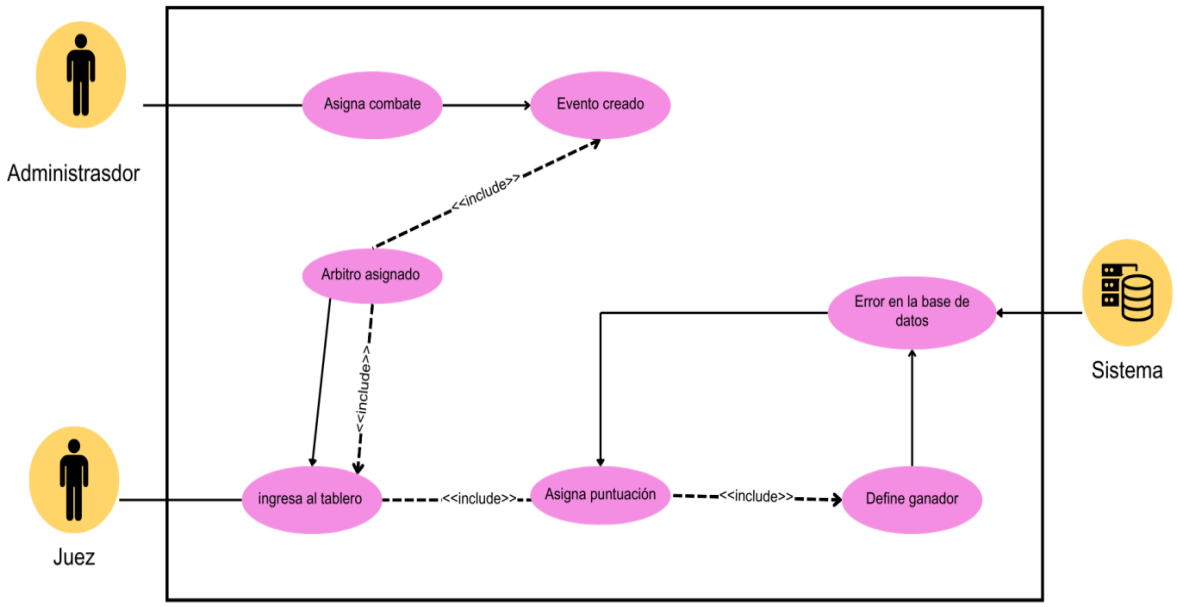
Diagrama de caso de uso

1. Caso Típico



2. Caso Atípico

Diagrama caso de uso atípico.
Error en el puntaje



Plantillas

Planilla (Caso de uso Atípico)

ombre del caso de uso:	Manejo de error en puntuación
Actor(es):	Administrador, Árbitro, Sistema (actor no humano)
Precondión:	El combate debe estar creado y asignado a un árbitro.
Flujo Principal:	1. Administrador asigna combate y árbitro. 2. Árbitro ingresa al tablero. 3. Asigna puntuación. 4. Sistema valida y guarda datos.
Flujo alternativo:	***Error en puntuación:*** - Sistema detecta inconsistencia (ej: datos corruptos). - Notifica al árbitro y administrador. - Solicita reintento o corrección manual.
Poscondición:	- Si hay error: Puntuación queda pendiente. - Si no: Ganador queda registrado.
Frecuencia:	Baja (ocurre solo por fallos técnicos).
Importancia:	Alta (afecta resultados oficiales del evento).

Planilla (Caso de uso Típico)

ombre del caso de uso:	Gestión de Evento Deportivo
Actor(es):	Mini Admin, Juez
Precondión:	- El usuario (Mini Admin) debe estar autenticado. - Datos básicos del evento disponibles.
Flujo Principal:	1. Mini Admin autoriza evento (logotipo externo). 2. Registra participantes y configuración. 3. Monitorea correcciones. 4. Juez selecciona ganadores.
Flujo alternativo:	***Datos incompletos:*** - Sistema solicita información faltante. ***Empate:*** - Juez aplica criterios de desempate.
Poscondición:	Evento queda registrado con ganadores definidos.
Frecuencia:	Alta (diaria/semanal, según eventos programados).
Importancia:	Media-Alta (esencial para operaciones rutinarias

Diagrama de estado

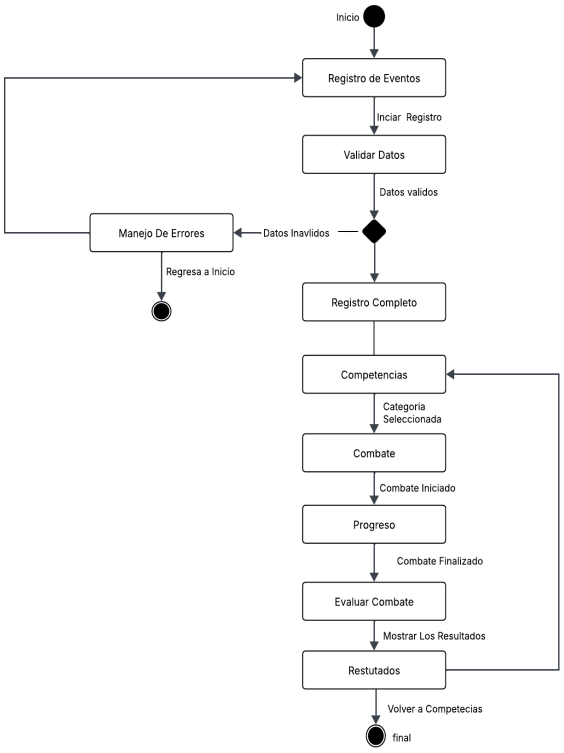


Diagrama de secuencia

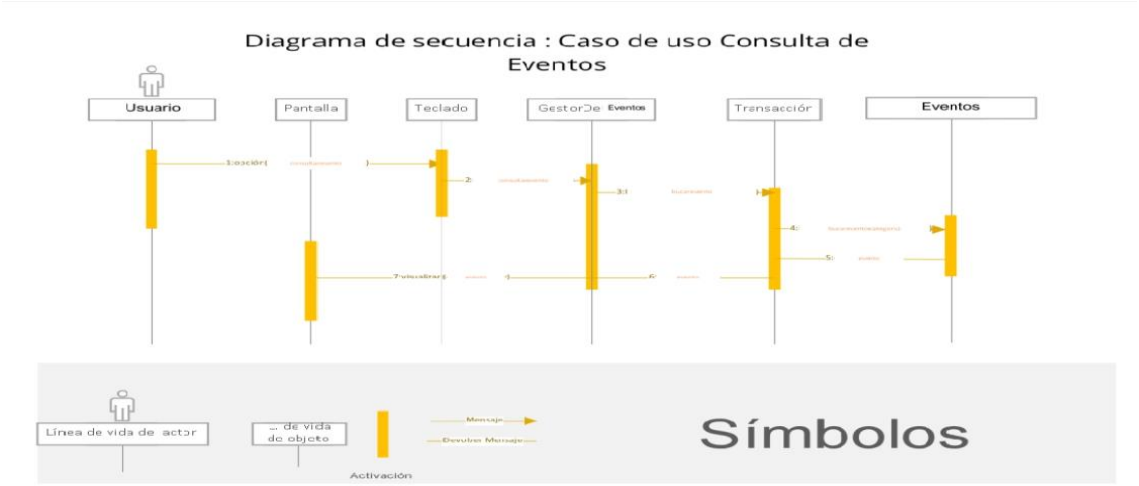


Diagrama de actividades

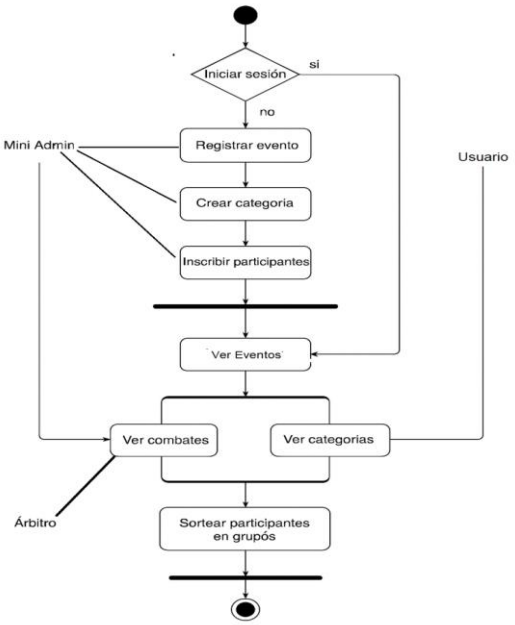
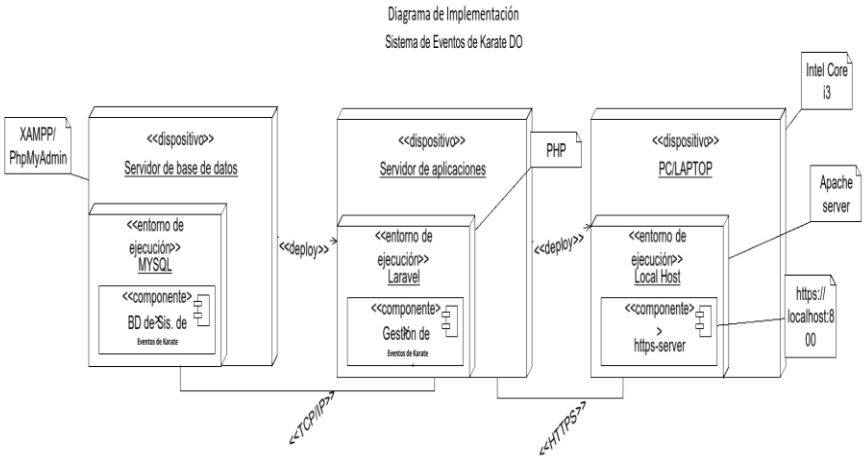
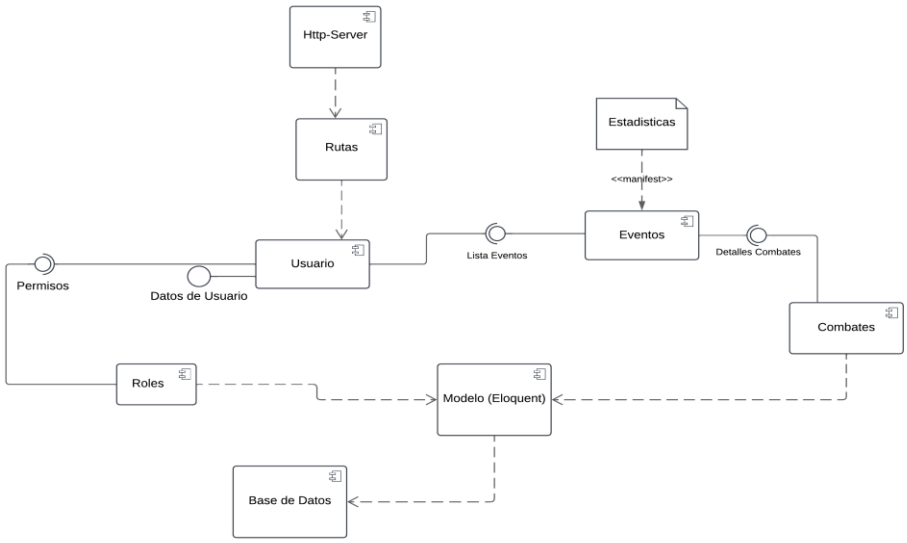


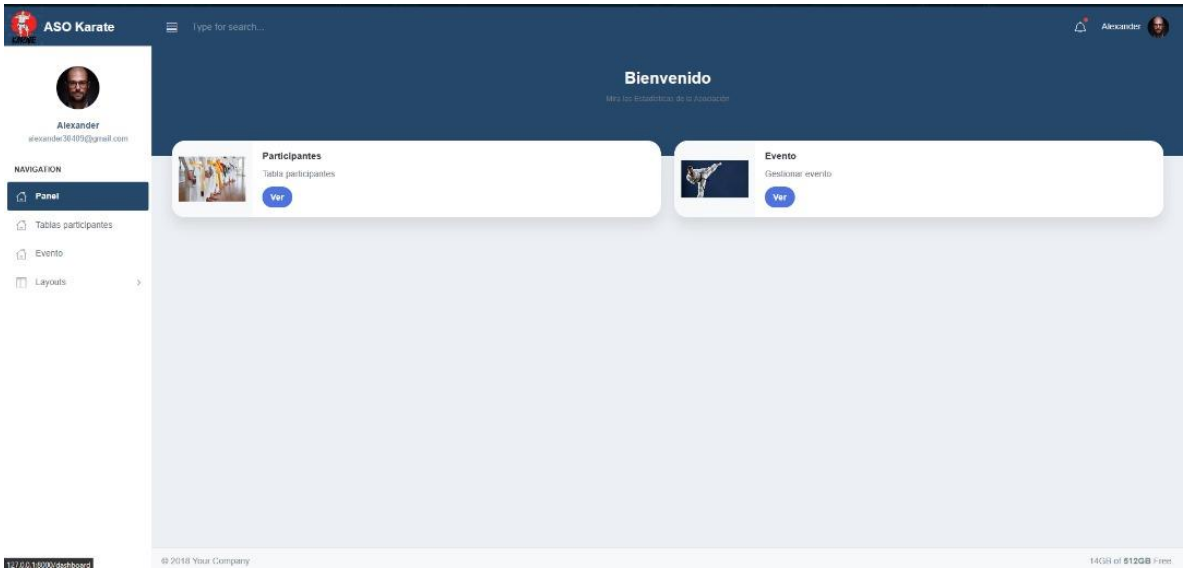
Diagrama De Implementación



Componentes



Interfaz Grafica



Home Sobre nosotros Servicios Contacto Soy Arbitro Iniciar sesion

Sobre Nosotros

Nuestro dojo se especializa en preparar competidores para torneos de karate, con entrenamiento enfocado en **kata, kumite y condición física de alto rendimiento**. Ofrecemos coaching personalizado, simulacros de competencia y estrategias para destacar en eventos locales, nacionales e internacionales.

¿Quieres competir? ¡Únete a nuestro equipo y lleva tu karate al siguiente nivel!

Preparación para torneos

Entrenamiento especializado en reglamentos de competición, puntajes y técnicas efectivas para ganar en torneos de kata y kumite.

Análisis de desempeño

Evaluación técnica y táctica con grabación en video para corregir detalles y maximizar tu potencial competitivo.

Participación en eventos

Te acompañamos a competencias y conectamos con organizadores para que tengas oportunidades en circuitos reconocidos.

Bienvenido de Vuelta

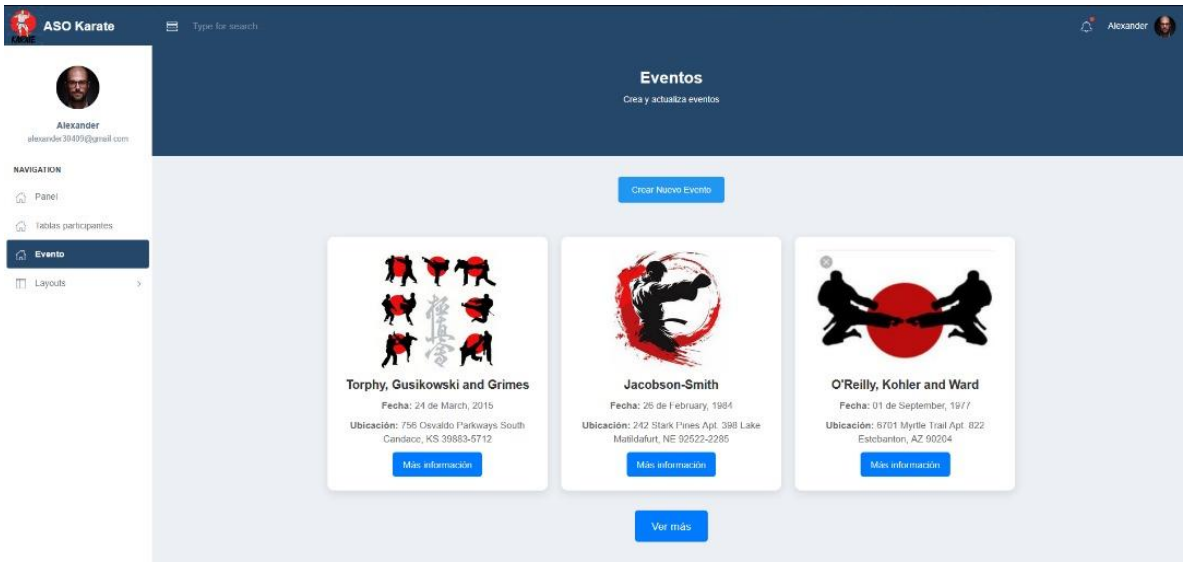
Mira las Estadísticas de la Asociacion

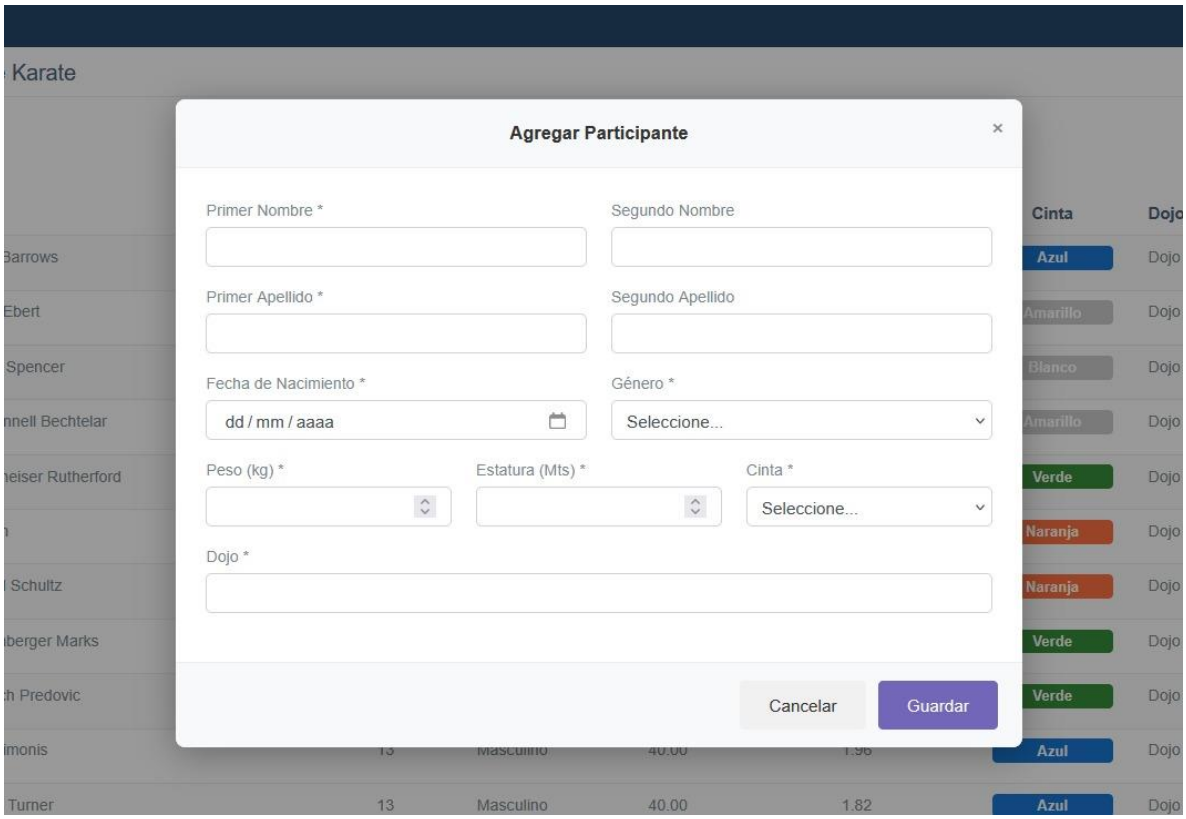
Participantes de Karate

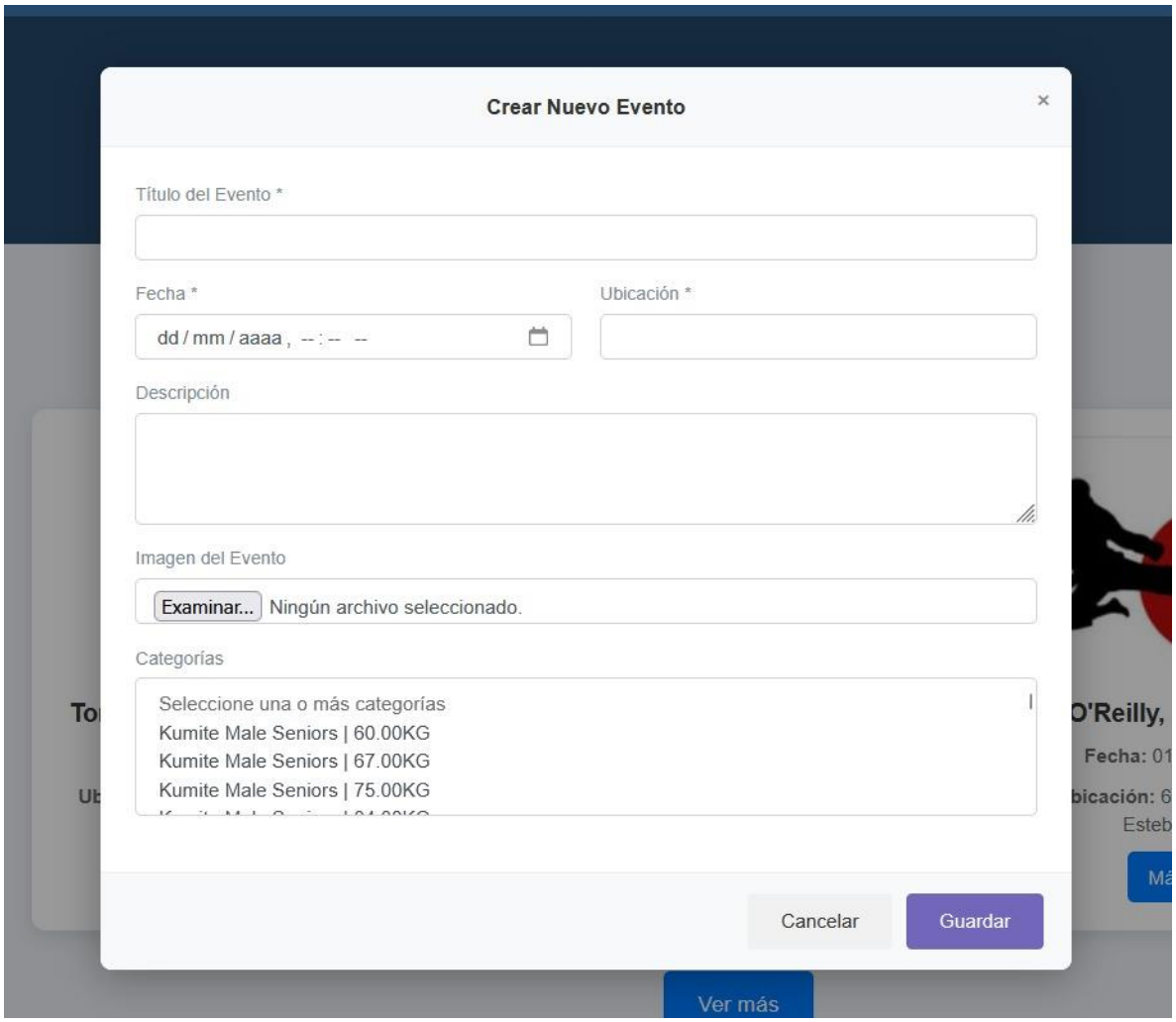
Agregar

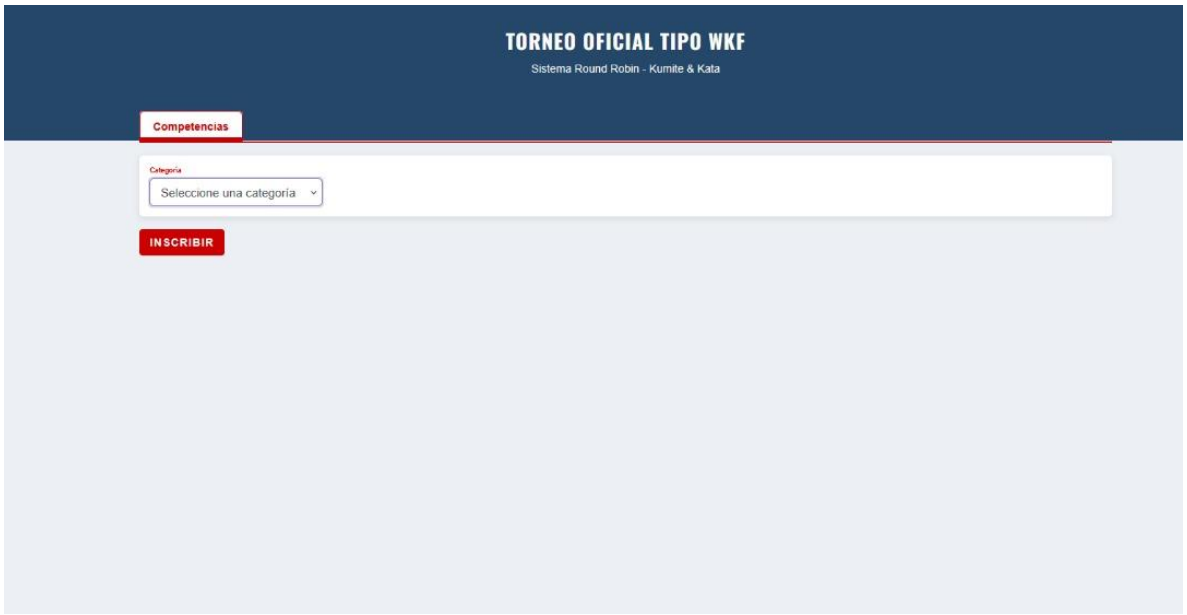
Buscar...

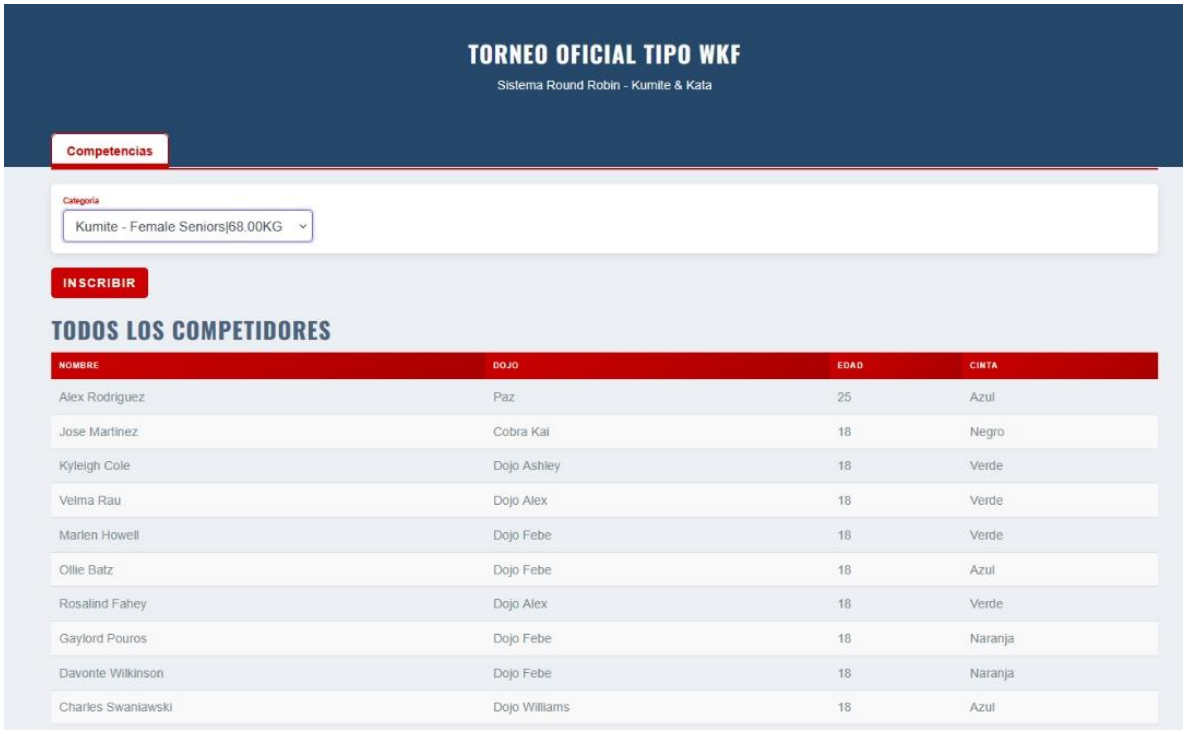
Nombre Completo	Edad	Género	Peso (kg)	Estatura (Mts)	Cinta	Dojo	Acciones
Kyla Wilton Heaney Barrows	13	Masculino	40.00	1.75	Azul	Dojo Shenn	
Cyril Lorna Stroman Ebert	13	Masculino	40.00	1.63	Amarillo	Dojo Alex	
Andy Talia DuBuque Spencer	13	Masculino	40.00	1.95	Blanco	Dojo Alex	
Elvera Maximo O'Connell Bechtelar	13	Masculino	40.00	1.65	Amarillo	Dojo Ashley	
Jaycee Monica Wintheiser Rutherford	13	Masculino	40.00	1.87	Verde	Dojo Williams	
Lavon Alex Kub Klein	13	Masculino	40.00	1.71	Naranja	Dojo Ashley	
Jarret Kayleigh Hand Schultz	13	Masculino	40.00	1.84	Naranja	Dojo Williams	
Mina Yesenia Schamberger Marks	13	Masculino	40.00	1.76	Verde	Dojo Ashley	
Jordon Coleman Koch Predovic	13	Masculino	40.00	1.98	Verde	Dojo Caraqueño	
Kyla Dexter Crona Simonis	13	Masculino	40.00	1.96	Azul	Dojo Williams	
Lisette Toy Weissnat Turner	13	Masculino	40.00	1.82	Azul	Dojo Alex	
Dayton Thalia Nikolaus Adams	13	Masculino	40.00	1.79	Verde	Dojo Caraqueño	
Jackson Nat Moen Hermann	13	Masculino	40.00	1.93	Azul	Dojo Caraqueño	

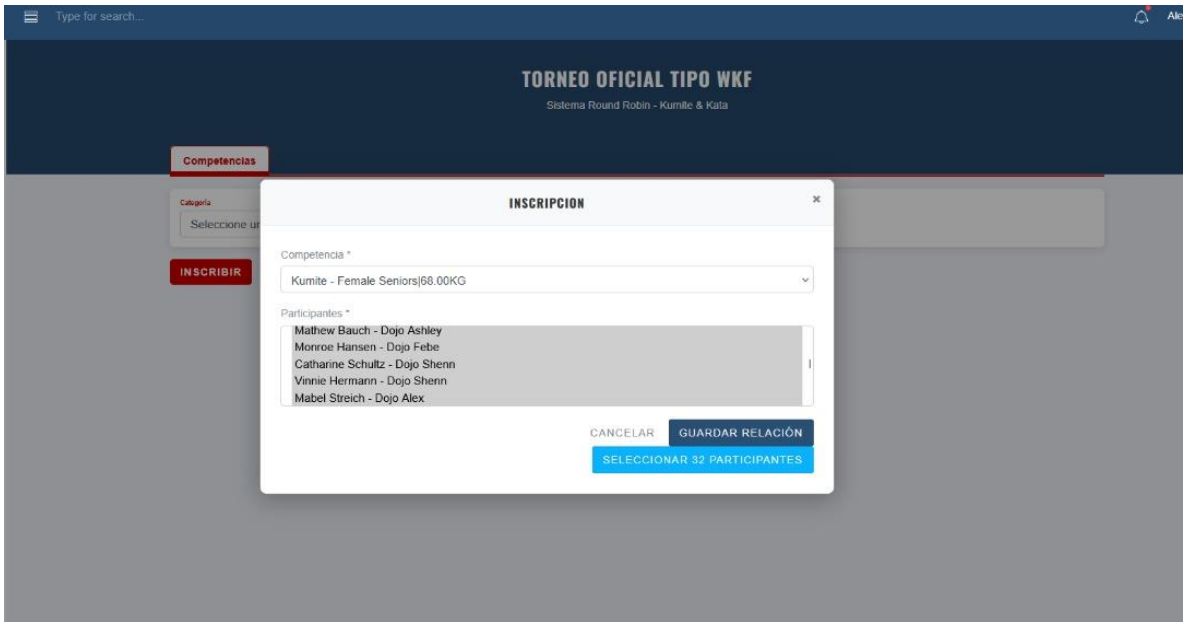

ASO Karate
Type for search
Alexander
Alexander
alexander30409@gmail.com
NAVIGATION
Panel
Tablas participantes
Evento
Layouts
Eventos
Crea y actualiza eventos
Crear Nuevo Evento
Torphy, Gusikowski and Grimes
Fecha: 24 de March, 2015
Ubicación: 756 Osvaldo Parkways South Candace, KS 39883-5712
Más información
Jacobson-Smith
Fecha: 26 de February, 1984
Ubicación: 242 Stark Pines Apt. 398 Lake Matildafurt, NE 92622-2285
Más información
O'Reilly, Kohler and Ward
Fecha: 01 de September, 1977
Ubicación: 6701 Myrtle Trail Apt. 822 Estebanton, AZ 90204
Más información
Ver más


Karate
Agregar Participante
Primer Nombre *
Segundo Nombre
Primer Apellido *
Segundo Apellido
Fecha de Nacimiento *
dd / mm / aaaa
Género *
Seleccione...
Peso (kg) *
Estatura (Mts) *
Cinta *
Seleccione...
Dojo *
Cancelar
Guardar
Cinta
Dojo
Azul
Amarillo
Blanco
Verde
Naranja
Turner
13
Masculino
40.00
1.82


Crear Nuevo Evento
Título del Evento *
Fecha *
dd / mm / aaaa , -- : -- --
Ubicación *
Descripción
Imagen del Evento
Examinar...
Ningún archivo seleccionado.
Categorías
Seleccione una o más categorías
Kumite Male Seniors | 60.00KG
Kumite Male Seniors | 67.00KG
Kumite Male Seniors | 75.00KG
Cancelar
Guardar
Ver más


TORNEO OFICIAL TIPO WKF
Sistema Round Robin - Kumite & Kata
Competencias
Categoría
Seleccione una categoría
INSCRIBIR

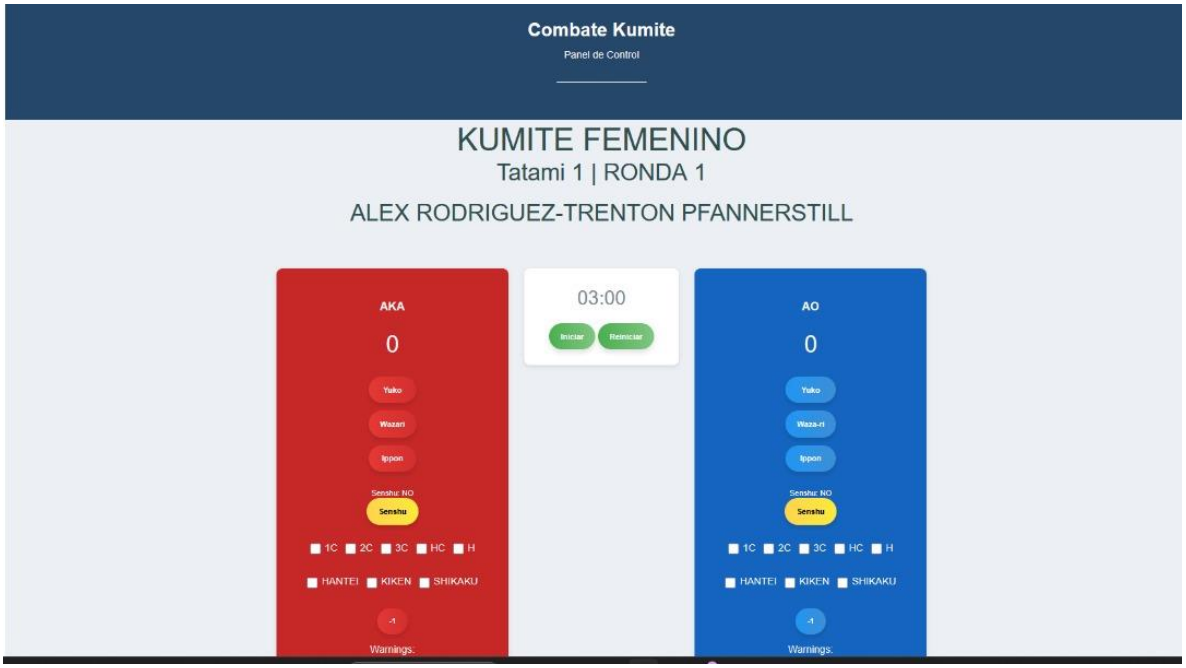

TORNEO OFICIAL TIPO WKF
Sistema Round Robin - Kumite & Kata
Competencias
Categoría
Kumite - Female Seniors|68.00KG
INSCRIBIR
TODOS LOS COMPETIDORES
NOMBRE
DOJO
EDAD
CINTA
Alex Rodriguez
Paz
25
Azul
Jose Martinez
Cobra Kai
18
Negro
Kyleigh Cole
Dojo Ashley
18
Verde
Velma Rau
Dojo Alex
18
Verde
Marlen Howell
Dojo Febe
18
Verde
Ollie Batz
Dojo Febe
18
Azul
Rosalind Fahey
Dojo Alex
18
Verde
Gaylord Pouros
Dojo Febe
18
Naranja
Davonte Wilkinson
Dojo Febe
18
Naranja
Charles Swaniawski
Dojo Williams
18
Azul

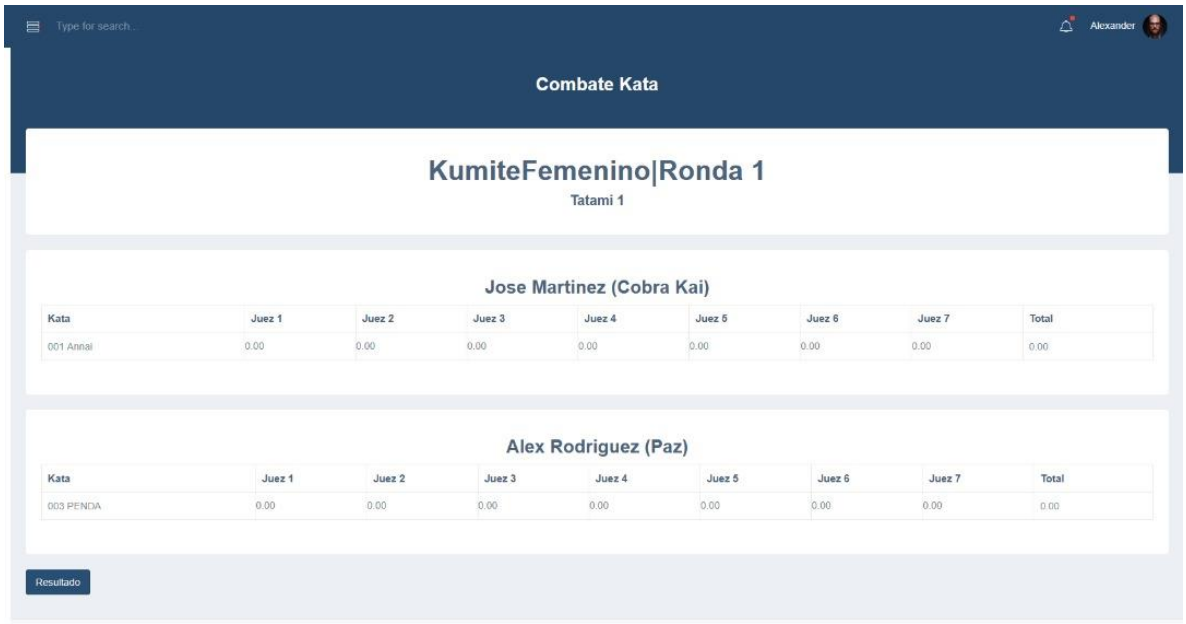


Erick Bartoletti	Dojo Shenn	18	Azul
Frida Kerluke	Dojo Williams	18	Azul
Sandra Jakubowski	Dojo Alex	18	Azul
Jerad Sanford	Dojo Shenn	18	Naranja
Gay Cole	Dojo Alex	18	Naranja
Kelley Kunde	Dojo Shenn	18	Blanco
Trenton Pfannerstill	Dojo Caraqueño	18	Amarillo
Raymond Schowalter	Dojo Ashley	18	Amarillo
Xander Kemmer	Dojo Ashley	18	Verde
Catherine Dickens	Dojo Ashley	18	Naranja
Brandyn Thompson	Dojo Shenn	18	Blanco
Mikayla Cummerata	Dojo Shenn	18	Verde
Ashlynn Kuvalis	Dojo Caraqueño	18	Azul
Brant Schuster	Dojo Shenn	18	Verde
Mathew Bauch	Dojo Ashley	18	Naranja
Monroe Hansen	Dojo Febe	18	Blanco
Catharine Schultz	Dojo Shenn	18	Azul
Vinnie Hermann	Dojo Shenn	18	Naranja
Mabel Streich	Dojo Alex	18	Naranja

SORTEAR


COMBATE KUMITE
EN VIVO
PAZ
RODRIGUEZ
ALEX
DOJO CARAQUEñO
PFANNERSTILL
TRENTON
0
0
TATAMI 1
F | 68 KG
26:15
GRUPO 1
RONDA 1

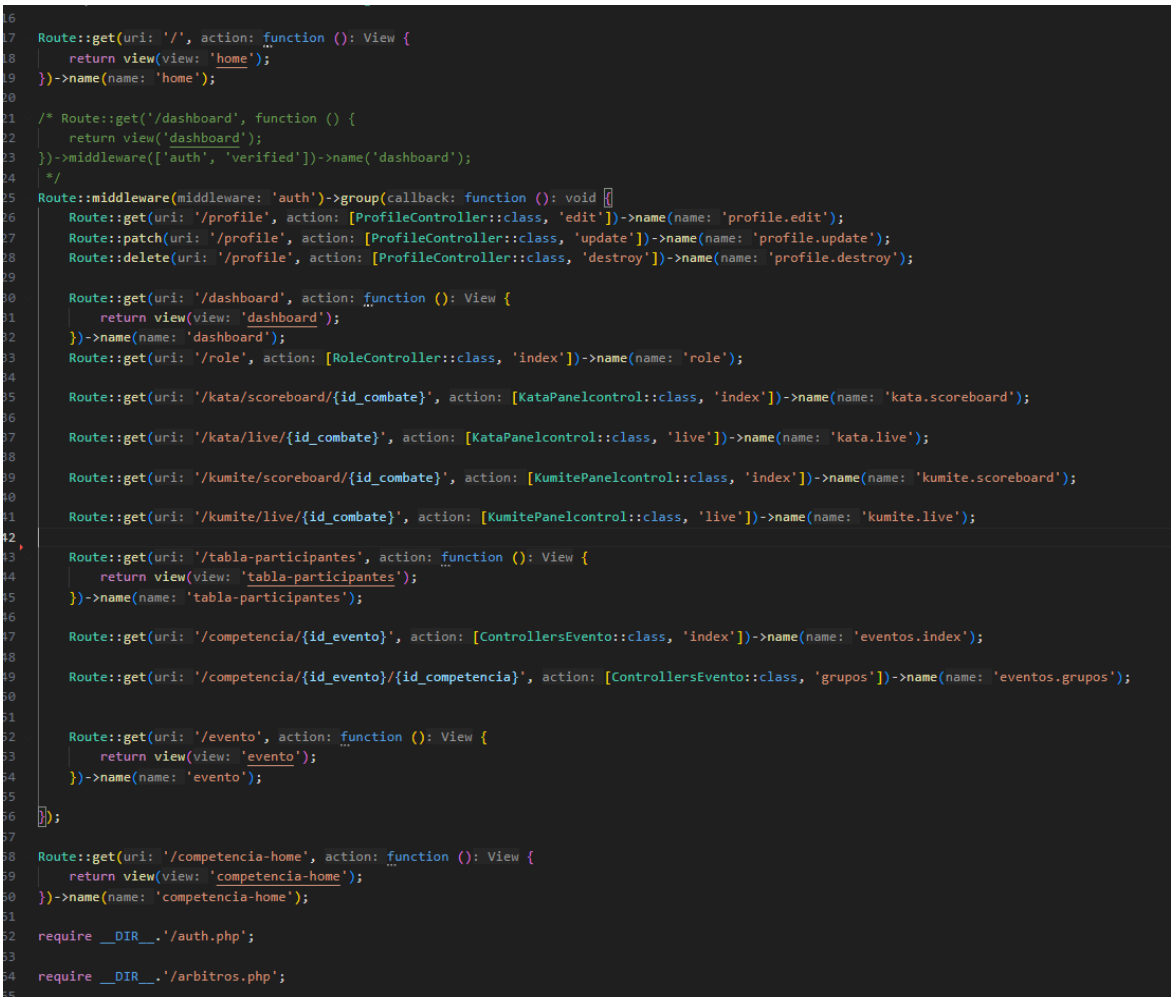

Combate Kumite
Panel de Control
KUMITE FEMENINO
Tatami 1 | RONDA 1
ALEX RODRIGUEZ-TRENTON PFANNERSTILL
AKA
0
Yuko
Wazari
Ippon
Senshu: NO
Senshu
1C 2C 3C HC H
HANTEI KIKEN SHIKAKU
-1
Warnings:
03:00
Iniciar
Reiniciar
AO
0
Yuko
Waza-ri
Ippon
Senshu: NO
Senshu
1C 2C 3C HC H
HANTEI KIKEN SHIKAKU
-1
Warnings:


Type for search...
Alexander
Combate Kata
KumiteFemenino|Ronda 1
Tatami 1
Jose Martinez (Cobra Kai)
Kata
Juez 1
Juez 2
Juez 3
Juez 4
Juez 5
Juez 6
Juez 7
Total
001 Annai
0.00
Alex Rodriguez (Paz)
003 PENDA
Resultado


EN VIVO
Tatami 1
KUMITE FEMENINO | Ronda 1
JOSE MARTINEZ
COBRA KAI
CINTURÓN: NEGRO
001 ANNAI
Performance
KUMITE FEMENINO | Ronda 1
ALEX RODRIGUEZ
PAZ
CINTURÓN: AZUL
003 PENDA

Archivo: web.php

Descripción: El archivo web.php define las rutas principales de la aplicación web en Laravel, especificando cómo las solicitudes HTTP se asignan a controladores o vistas. Este archivo organiza las rutas en grupos, como aquellas protegidas por el middleware auth, que requieren autenticación para acceder a funcionalidades como la edición de perfiles, el panel de control, y la gestión de roles. También incluye rutas específicas para manejar eventos y competencias de karate, como los tableros de puntuación y vistas en vivo para kata y kumite. Además, se definen rutas para vistas estáticas como tabla-participantes y competencia-home, y se cargan archivos adicionales de rutas (auth.php y arbitros.php) para modularidad.



```php
Route::get(uri: '/', action: function (): View {
    return view(view: 'home');
})->name(name: 'home');

/* Route::get('/dashboard', function () {
    return view('dashboard');
})->middleware(['auth', 'verified'])->name('dashboard');
 */
Route::middleware(middleware: 'auth')->group(callback: function (): void {
    Route::get(uri: '/profile', action: [ProfileController::class, 'edit'])->name(name: 'profile.edit');
    Route::patch(uri: '/profile', action: [ProfileController::class, 'update'])->name(name: 'profile.update');
    Route::delete(uri: '/profile', action: [ProfileController::class, 'destroy'])->name(name: 'profile.destroy');

    Route::get(uri: '/dashboard', action: function (): View {
        return view(view: 'dashboard');
    })->name(name: 'dashboard');
    Route::get(uri: '/role', action: [RoleController::class, 'index'])->name(name: 'role');

    Route::get(uri: '/kata/scoreboard/{id_combate}', action: [KataPanelcontrol::class, 'index'])->name(name: 'kata.scoreboard');

    Route::get(uri: '/kata/live/{id_combate}', action: [KataPanelcontrol::class, 'live'])->name(name: 'kata.live');

    Route::get(uri: '/kumite/scoreboard/{id_combate}', action: [KumitePanelcontrol::class, 'index'])->name(name: 'kumite.scoreboard');

    Route::get(uri: '/kumite/live/{id_combate}', action: [KumitePanelcontrol::class, 'live'])->name(name: 'kumite.live');

    Route::get(uri: '/tabla-participantes', action: function (): View {
        return view(view: 'tabla-participantes');
    })->name(name: 'tabla-participantes');

    Route::get(uri: '/competencia/{id_evento}', action: [ControllersEvento::class, 'index'])->name(name: 'eventos.index');

    Route::get(uri: '/competencia/{id_evento}/{id_competencia}', action: [ControllersEvento::class, 'grupos'])->name(name: 'eventos.grupos');


    Route::get(uri: '/evento', action: function (): View {
        return view(view: 'evento');
    })->name(name: 'evento');

});

Route::get(uri: '/competencia-home', action: function (): View {
    return view(view: 'competencia-home');
})->name(name: 'competencia-home');

require __DIR__.'/auth.php';

require __DIR__.'/arbitros.php';
```

Carpeta: App/Models

Descripción: La carpeta Models en una aplicación Laravel contiene las clases de modelo que representan las tablas de la base de datos y encapsulan la lógica relacionada con los datos. Estos modelos permiten interactuar con la base de datos de manera sencilla utilizando Eloquent ORM, proporcionando métodos para realizar consultas, relaciones entre tablas (como belongsToMany, hasMany, etc.), y definir atributos personalizados o formateados. En resumen, esta carpeta organiza y centraliza la lógica de acceso y manipulación de datos de la aplicación.

```
19 references | 0 implementations
class competencia extends Model
{
    0 references
    protected $fillable = ['id_evento','id_categoria','disciplina'];
    0 references
    protected $table='competencias';

    0 references | 0 overrides
    public function evento():BelongsTo{
        return $this->belongsTo(related: evento::class,foreignKey: 'id_evento');
    }
    0 references | 0 overrides
    public function categoria():BelongsTo{
        return $this->belongsTo(related: categoria::class,foreignKey: 'id_categoria');
    }
    //https://laravel.com/docs/12.x/eloquent-relationships

    0 references | 0 overrides
    public function combates(): HasMany{
        return $this->hasMany(related: Combate::class,foreignKey: 'id_competencia');
    }

    0 references | 0 overrides
    public function grupos(): HasMany{
        return $this->hasMany(related: Grupo::class,foreignKey: 'id_competencia');
    }

    1 reference | 0 overrides
    public function participantes(): BelongsToMany{
        return $this->belongsToMany(related: participantes::class,table: 'competencias_participantes',foreignPivotKey: 'id_competencia',relatedPiv…'id_participante');
    }
}
```

Archivo: competencia.php

Carpeta: app/Http/Controller/Auth

Descripción: La carpeta Auth en una aplicación Laravel contiene los controladores responsables de gestionar la autenticación de usuarios. Estos controladores manejan funcionalidades como el inicio de sesión, registro, cierre de sesión, restablecimiento de contraseñas y verificación de correos electrónicos. Laravel incluye controladores predeterminados en esta carpeta para simplificar la implementación de estas características, permitiendo personalizarlas según las necesidades de la aplicación. En resumen, esta carpeta centraliza la lógica relacionada con la autenticación y la seguridad de acceso de los usuarios.

```php
4 references | 0 implementations
class AuthenticatedSessionController extends Controller
{
    /**
     * Display the login view.
     */
    1 reference | 0 overrides
    public function create(): View
    {
        return view(view: 'auth.login');
    }

    /**
     * Handle an incoming authentication request.
     */
    1 reference | 0 overrides
    public function store(LoginRequest $request): RedirectResponse
    {
        $request->authenticate();

        $request->session()->regenerate();

        return redirect()->intended(default: route(name: 'dashboard', absolute: false));
    }

    /**
     * Destroy an authenticated session.
     */
    1 reference | 0 overrides
    public function destroy(Request $request): RedirectResponse
    {
        Auth::guard(name: 'web')->logout();

        $request->session()->invalidate();

        $request->session()->regenerateToken();

        return redirect(to: '/');
    }
}
```

Archivo:AuthenticatedSessionController.php

Archivo: app/Http/Controller/RoleController.php

Descripción: El archivo RoleController.php define la lógica para gestionar el acceso a la vista de roles en la aplicación. Su método index verifica si el usuario autenticado tiene el rol de administrador (role === 'admin'). Si es así, retorna la vista role con información del usuario; de lo contrario, redirige al panel de control (dashboard) con un mensaje de error indicando que no tiene acceso. Este controlador asegura que solo usuarios con permisos específicos puedan acceder a ciertas funcionalidades relacionadas con roles.

```php
2 references | 0 implementations
class RoleController extends Controller
{
    1 reference | 0 overrides
    public function index(Request $request): View|RedirectResponse
    {
        if( Auth::check() && Auth::user()->role === 'admin'){
            return view(view: 'role', data: [
                'user' => $request->user(),
            ]);
        }else{
            return Redirect::route(route: 'dashboard')->with(key: 'error', value: 'You do not have access to this page.');
        }
    }
}
```

Archivo: app/Http/Controller/evento.php

Descripción: El archivo evento.php contiene la lógica para gestionar las vistas relacionadas con eventos y competencias en la aplicación. El método index busca un evento específico por su ID, obtiene sus categorías asociadas y retorna la vista competencia con estos datos. Por otro lado, el método grupos busca una competencia específica por su ID, obtiene los grupos asociados y retorna la vista grupos con esta información. Este controlador facilita la interacción entre los modelos de eventos y competencias y las vistas correspondientes, permitiendo mostrar datos dinámicos al usuario.

```php
class evento extends Controller
{
    1 reference | 0 overrides
    public function index($id_evento): View
    {
        // Busca el evento por su ID
        $evento = ModelsEvento::findOrFail(id: $id_evento);
        $categorias = $evento->categorias;

        // Retorna la vista con los datos del evento
        return view(view: 'competencia', data: compact(var_name: 'evento', var_names: 'categorias'));
    }

    1 reference | 0 overrides
    public function grupos($id_evento,$id_competencia): View
    {
        // Busca el evento por su ID
        $competencia = competencia::findOrFail(id: $id_competencia);
        $grupos = $competencia->grupos;

        // Retorna la vista con los datos del evento
        return view(view: 'grupos', data: compact(var_name: 'competencia', var_names: 'grupos'));
    }

}
```

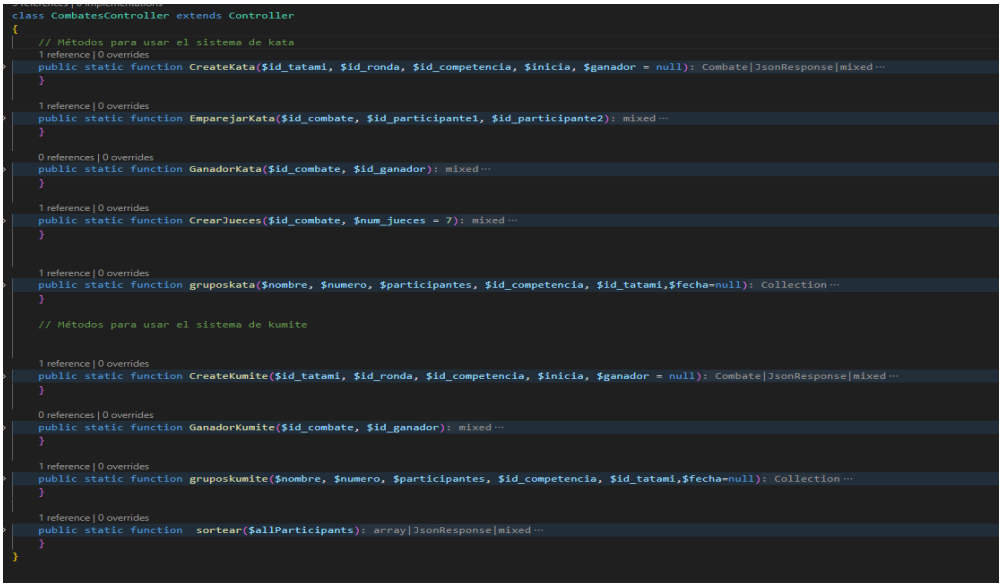
Archivo: App/Http/Controller/CombateController.php

Descripción:

1. **CreateKata**
    - Crea un combate de kata en la base de datos.
    - Valida los parámetros de entrada y verifica que la competencia sea de kata.
    - Establece el estado del combate según la fecha de inicio y guarda el ganador si se proporciona.
2. **EmparejarKata**
    - Asigna participantes o equipos a un combate de kata.
    - Verifica si el combate y los participantes existen.
    - Crea equipos de kata si no existen y los asocia al combate.
    - Genera presentaciones de kata para los equipos.
3. **GanadorKata**
    - Actualiza el ganador de un combate de kata.
    - Verifica que el combate exista y que sea de kata antes de guardar el ganador.
4. **CrearJueces**
    - Crea jueces para un combate de kata.
    - Verifica que el combate exista y que sea de kata.
    - Asocia jueces y sus puntuaciones al combate.
5. **gruposkata**
    - Crea un grupo de kata y lo asocia a una competencia.
    - Asigna participantes o equipos al grupo y genera rondas y combates entre ellos.
    - Crea jueces y emparejamientos para los combates generados.
6. **CreateKumite**
    - Crea un combate de kumite en la base de datos.
    - Valida los parámetros de entrada y verifica que la competencia sea de kumite.
    - Establece el estado del combate según la fecha de inicio y guarda el ganador si se proporciona.
7. **GanadorKumite**

- Actualiza el ganador de un combate de kumite.
- Verifica que el combate exista y que sea de kumite antes de guardar el ganador.

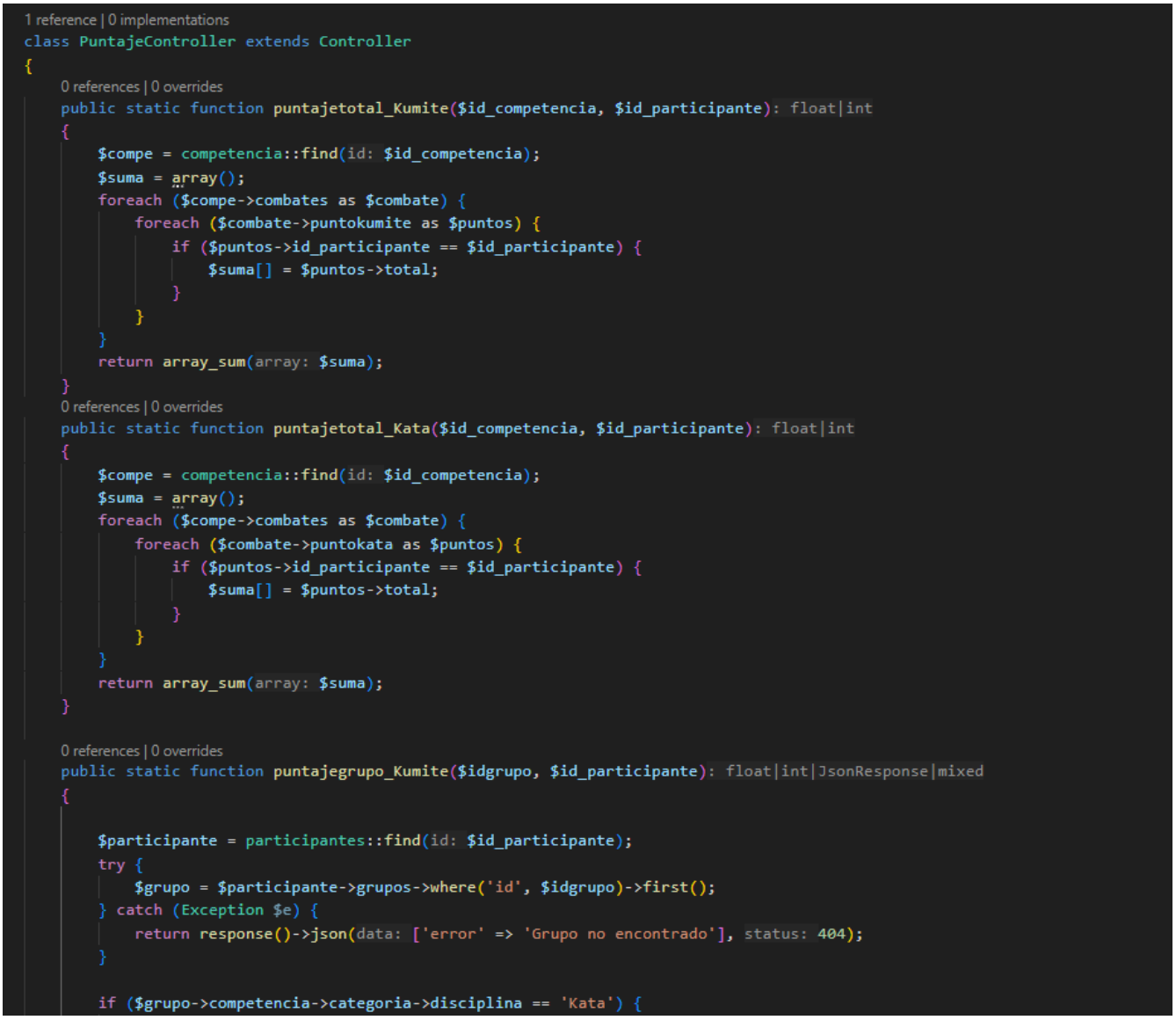
8. **gruposkumite**
    - Crea un grupo de kumite y lo asocia a una competencia.
    - Asigna participantes al grupo y genera rondas y combates entre ellos.
    - Asocia participantes y puntuaciones a los combates generados.
9. **sortear**
    - Divide a 32 participantes en 8 grupos de manera aleatoria.
    - Verifica que el número de participantes sea exactamente 32 antes de realizar el sorteo.



```
class CombatesController extends Controller
{
    // Métodos para usar el sistema de kata
    1 reference | 0 overrides
    public static function CreateKata($id_tatami, $id_ronda, $id_competencia, $inicia, $ganador = null): Combate|JsonResponse|mixed ···
    }

    1 reference | 0 overrides
    public static function EmparejarKata($id_combate, $id_participante1, $id_participante2): mixed ···
    }

    0 references | 0 overrides
    public static function GanadorKata($id_combate, $id_ganador): mixed ···
    }

    1 reference | 0 overrides
    public static function CrearJueces($id_combate, $num_jueces = 7): mixed ···
    }


    1 reference | 0 overrides
    public static function gruposkata($nombre, $numero, $participantes, $id_competencia, $id_tatami,$fecha=null): Collection ···
    }

    // Métodos para usar el sistema de kumite


    1 reference | 0 overrides
    public static function CreateKumite($id_tatami, $id_ronda, $id_competencia, $inicia, $ganador = null): Combate|JsonResponse|mixed ···
    }

    0 references | 0 overrides
    public static function GanadorKumite($id_combate, $id_ganador): mixed ···
    }

    1 reference | 0 overrides
    public static function gruposkumite($nombre, $numero, $participantes, $id_competencia, $id_tatami,$fecha=null): Collection ···
    }

    1 reference | 0 overrides
    public static function  sortear($allParticipants): array|JsonResponse|mixed ···
    }
}
```

Archivo: app/Http/Controller/PuntajeController.php

Descripción: El archivo PuntajeController.php contiene métodos para calcular los puntajes totales y grupales de los participantes en competencias de kumite y kata. Incluye funciones como puntajetotal_Kumite y puntajetotal_Kata, que suman los puntos obtenidos por un participante en todos los combates de una competencia, y puntajegrupo_Kumite y puntajegrupo_Kata, que calculan los puntos acumulados por un participante dentro de un grupo específico. Además, valida que los datos sean correctos y que las disciplinas coincidan con el tipo de competencia, retornando errores en caso de inconsistencias.



```
1 reference | 0 implementations
class PuntajeController extends Controller
{
    0 references | 0 overrides
    public static function puntajetotal_Kumite($id_competencia, $id_participante): float|int
    {
        $compe = competencia::find(id: $id_competencia);
        $suma = array();
        foreach ($compe->combates as $combate) {
            foreach ($combate->puntokumite as $puntos) {
                if ($puntos->id_participante == $id_participante) {
                    $suma[] = $puntos->total;
                }
            }
        }
        return array_sum(array: $suma);
    }
    0 references | 0 overrides
    public static function puntajetotal_Kata($id_competencia, $id_participante): float|int
    {
        $compe = competencia::find(id: $id_competencia);
        $suma = array();
        foreach ($compe->combates as $combate) {
            foreach ($combate->puntokata as $puntos) {
                if ($puntos->id_participante == $id_participante) {
                    $suma[] = $puntos->total;
                }
            }
        }
        return array_sum(array: $suma);
    }

    0 references | 0 overrides
    public static function puntajegrupo_Kumite($idgrupo, $id_participante): float|int|JsonResponse|mixed
    {

        $participante = participantes::find(id: $id_participante);
        try {
            $grupo = $participante->grupos->where('id', $idgrupo)->first();
        } catch (Exception $e) {
            return response()->json(data: ['error' => 'Grupo no encontrado'], status: 404);
        }

        if ($grupo->competencia->categoria->disciplina == 'Kata') {
```

Archivo: app/Http/Controller/kumite/panelcontrol.php

Descripción: El archivo panelcontrol.php (similar al de kata) probablemente gestiona las vistas relacionadas con el panel de control y la visualización en vivo de los combates de kumite. Aunque no se muestra el contenido exacto, su función sería retornar vistas específicas, como un tablero de puntuación (Scoreboards.PanelKumite) y una vista en vivo (Scoreboards.kumitelive), pasando el identificador del combate como parámetro. Esto permite mostrar información dinámica sobre los combates de kumite en tiempo real o en un panel administrativo.

```php
class panelcontrol extends Controller
{
    1 reference | 0 overrides
    public function index($id_combate): View
    {
        $combate=Combate::find(id: $id_combate);
        $remaining = $combate->competencia->categoria->duracion*60;
        return view(view: 'Scoreboards.PanelKumite', data: ['id_combate'=>$id_combate,'remaining'=>$remaining]);
    }

    1 reference | 0 overrides
    public function live($id_combate): View
    {
        $combate=Combate::find(id: $id_combate);
        return view(view: 'Scoreboards.kumitelive', data: ['id_combate' => $id_combate]);
    }
}
```

Archivo: app/Http/Controller/kata/panelcontrol.php

Descripción: El archivo panelcontrol.php gestiona las vistas relacionadas con el panel de control y la visualización en vivo de los combates de kata. Contiene dos métodos: index, que retorna la vista del panel de control (Scoreboards.PanelKata) con el identificador del combate, y live, que muestra la vista en vivo del combate (Scoreboards.katalive) con el mismo identificador. Este controlador permite mostrar información específica y dinámica sobre los combates de kata en diferentes contextos, como administración o transmisión en tiempo real.

```php
class panelcontrol extends Controller
{
    1 reference | 0 overrides
    public function index($id_combate): View
    {
        return view(view: 'Scoreboards.PanelKata', data: ['id_combate' => $id_combate]);
    }
    1 reference | 0 overrides
    public function live($id_combate): View
    {
        return view(view: 'Scoreboards.katalive', data: ['id_combate' => $id_combate]);
    }
}
```

Carpeta: app/Http/Events

Descripción: Permite la transmisión en tiempo real de datos relacionados con combates de kata, kumite y el temporizador, utilizando canales de broadcasting personalizados según el identificador del combate. Esto facilita la sincronización dinámica de información entre el servidor y los clientes.

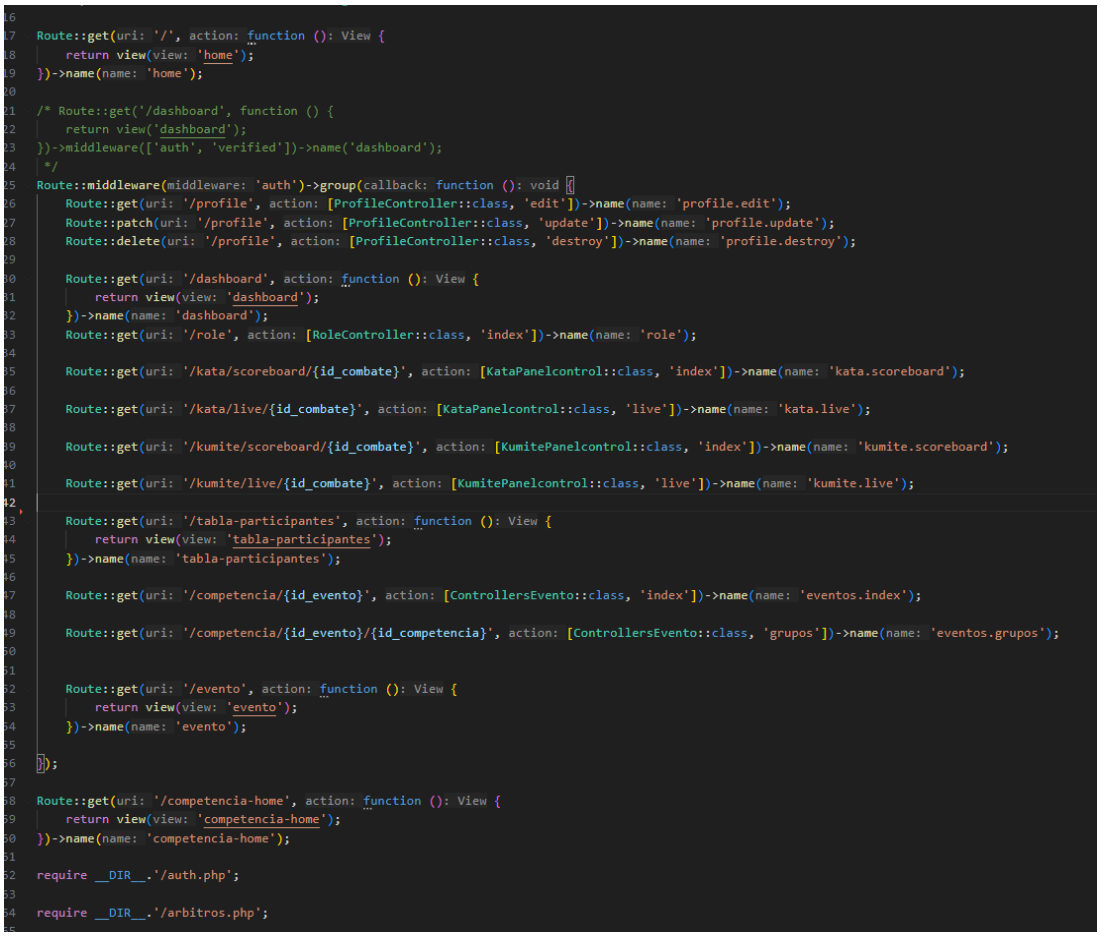
Estructura General de cada archivo:

1. **Propiedades**:
   - data: Contiene los datos que se transmitirán.
   - id_combate: Identificador del combate para personalizar el canal.
2. **Métodos**:
   - broadcastOn (): Transmite el evento en un canal público nombrado channel. {id_combate}.
   - broadcastAs (): Define el nombre del evento.
   - broadcastWith (): Envía los datos asociados al evento.

```
9 references | 0 implementations
class LiveKumite implements ShouldBroadcastNow
{
    use Dispatchable, InteractsWithSockets, SerializesModels;

    /**
     * Create a new event instance.
     */
    3 references
    public $data;
    2 references
    public $id_combate;
    0 references | 0 overrides
    public function __construct($data)
    {
        $this->data = $data;
        $this->id_combate = $data['id_combate'];
    }

    /**
     * Get the channels the event should broadcast on.
     *
     * @return \Illuminate\Broadcasting\Channel
     */
    0 references | 0 overrides
    public function broadcastOn(): array
    {
        return [
            new Channel(name: 'channel-Kumite.'. $this->id_combate), // Nombre del canal
        ];
    }
    /**
     * Datos que se enviarán con el evento.
     *
     * @return array
     */
    0 references | 0 overrides
    public function broadcastAs():string
    {
        return "livekumite"; // Nombre del evento que se enviará al canal
    }
    // Puedes personalizar los datos que se envían con el evento aquí
    // Por ejemplo, puedes enviar la variable $data que has pasado al constructor
    0 references | 0 overrides
    public function broadcastWith(): array
    {
        return [
            'data' => $this->data, // Enviar la variable como parte de los datos
        ];
    }
```

Archivo: web.php

Descripción: El archivo web.php define las rutas principales de la aplicación web en Laravel, especificando cómo las solicitudes HTTP se asignan a controladores o vistas. Este archivo organiza las rutas en grupos, como aquellas protegidas por el middleware auth, que requieren autenticación para acceder a funcionalidades como la edición de perfiles, el panel de control, y la gestión de roles. También incluye rutas específicas para manejar eventos y competencias de karate, como los tableros de puntuación y vistas en vivo para kata y kumite. Además, se definen rutas para vistas estáticas como tabla-participantes y competencia-home, y se cargan archivos adicionales de rutas (auth.php y arbitros.php) para modularidad.



```php

Route::get(uri: '/', action: function (): View {
    return view(view: 'home');
})->name(name: 'home');

/* Route::get('/dashboard', function () {
    return view('dashboard');
})->middleware(['auth', 'verified'])->name('dashboard');
 */
Route::middleware(middleware: 'auth')->group(callback: function (): void {
    Route::get(uri: '/profile', action: [ProfileController::class, 'edit'])->name(name: 'profile.edit');
    Route::patch(uri: '/profile', action: [ProfileController::class, 'update'])->name(name: 'profile.update');
    Route::delete(uri: '/profile', action: [ProfileController::class, 'destroy'])->name(name: 'profile.destroy');

    Route::get(uri: '/dashboard', action: function (): View {
        return view(view: 'dashboard');
    })->name(name: 'dashboard');
    Route::get(uri: '/role', action: [RoleController::class, 'index'])->name(name: 'role');

    Route::get(uri: '/kata/scoreboard/{id_combate}', action: [KataPanelcontrol::class, 'index'])->name(name: 'kata.scoreboard');

    Route::get(uri: '/kata/live/{id_combate}', action: [KataPanelcontrol::class, 'live'])->name(name: 'kata.live');

    Route::get(uri: '/kumite/scoreboard/{id_combate}', action: [KumitePanelcontrol::class, 'index'])->name(name: 'kumite.scoreboard');

    Route::get(uri: '/kumite/live/{id_combate}', action: [KumitePanelcontrol::class, 'live'])->name(name: 'kumite.live');

    Route::get(uri: '/tabla-participantes', action: function (): View {
        return view(view: 'tabla-participantes');
    })->name(name: 'tabla-participantes');

    Route::get(uri: '/competencia/{id_evento}', action: [ControllersEvento::class, 'index'])->name(name: 'eventos.index');

    Route::get(uri: '/competencia/{id_evento}/{id_competencia}', action: [ControllersEvento::class, 'grupos'])->name(name: 'eventos.grupos');


    Route::get(uri: '/evento', action: function (): View {
        return view(view: 'evento');
    })->name(name: 'evento');

});

Route::get(uri: '/competencia-home', action: function (): View {
    return view(view: 'competencia-home');
})->name(name: 'competencia-home');

require __DIR__.'/auth.php';

require __DIR__.'/arbitros.php';
```

Carpeta: App/Models

Descripción: La carpeta Models en una aplicación Laravel contiene las clases de modelo que representan las tablas de la base de datos y encapsulan la lógica relacionada con los datos. Estos modelos permiten interactuar con la base de datos de manera sencilla utilizando Eloquent ORM, proporcionando métodos para realizar consultas, relaciones entre tablas (como belongsToMany, hasMany, etc.), y definir atributos personalizados o formateados. En resumen, esta carpeta organiza y centraliza la lógica de acceso y manipulación de datos de la aplicación.

```php
19 references | 0 implementations
class competencia extends Model
{
    0 references
    protected $fillable = ['id_evento','id_categoria','disciplina'];
    0 references
    protected $table='competencias';

    0 references | 0 overrides
    public function evento():BelongsTo{
        return $this->belongsTo(related: evento::class,foreignKey: 'id_evento');
    }
    0 references | 0 overrides
    public function categoria():BelongsTo{
        return $this->belongsTo(related: categoria::class,foreignKey: 'id_categoria');
    }
    //https://laravel.com/docs/12.x/eloquent-relationships

    0 references | 0 overrides
    public function combates(): HasMany{
        return $this->hasMany(related: Combate::class,foreignKey: 'id_competencia');
    }

    0 references | 0 overrides
    public function grupos(): HasMany{
        return $this->hasMany(related: Grupo::class,foreignKey: 'id_competencia');
    }

    1 reference | 0 overrides
    public function participantes(): BelongsToMany{
        return $this->belongsToMany(related: participantes::class,table: 'competencias_participantes',foreignPivotKey: 'id_competencia',relatedPiv…'id_participante');
    }
}
```

Archivo: competencia.php

Carpeta: app/Http/Controller/Auth

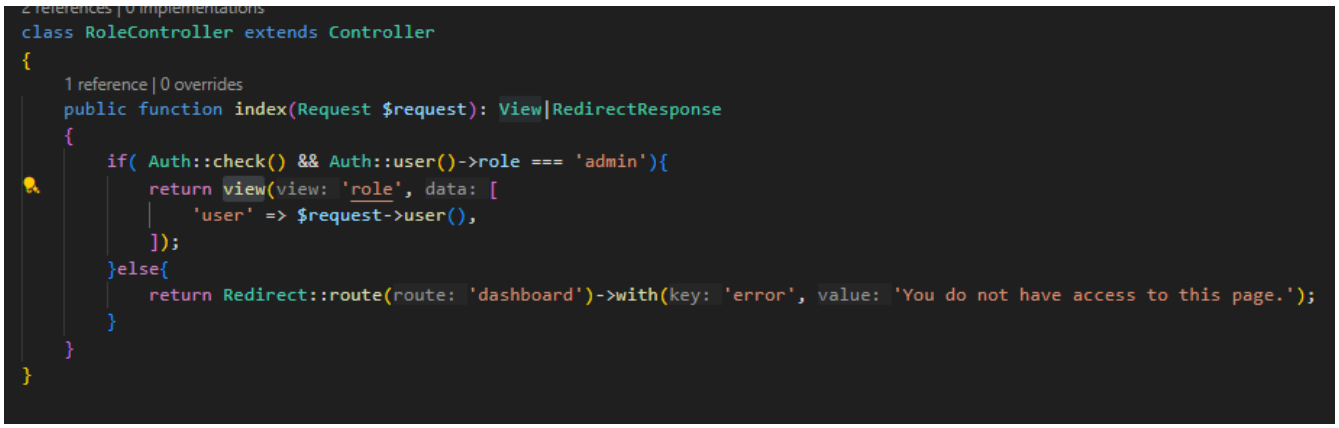
Descripción: La carpeta Auth en una aplicación Laravel contiene los controladores responsables de gestionar la autenticación de usuarios. Estos controladores manejan funcionalidades como el inicio de sesión, registro, cierre de sesión, restablecimiento de contraseñas y verificación de correos electrónicos. Laravel incluye controladores predeterminados en esta carpeta para simplificar la implementación de estas características, permitiendo personalizarlas según las necesidades de la aplicación. En resumen, esta carpeta centraliza la lógica relacionada con la autenticación y la seguridad de acceso de los usuarios.

```php
4 references | 0 implementations
class AuthenticatedSessionController extends Controller
{
    /**
     * Display the login view.
     */
    1 reference | 0 overrides
    public function create(): View
    {
        return view(view: 'auth.login');
    }

    /**
     * Handle an incoming authentication request.
     */
    1 reference | 0 overrides
    public function store(LoginRequest $request): RedirectResponse
    {
        $request->authenticate();

        $request->session()->regenerate();

        return redirect()->intended(default: route(name: 'dashboard', absolute: false));
    }

    /**
     * Destroy an authenticated session.
     */
    1 reference | 0 overrides
    public function destroy(Request $request): RedirectResponse
    {
        Auth::guard(name: 'web')->logout();

        $request->session()->invalidate();

        $request->session()->regenerateToken();

        return redirect(to: '/');
    }
}
```

Archivo: AuthenticatedSessionController.php

Archivo: app/Http/Controller/RoleController.php

Descripción: El archivo RoleController.php define la lógica para gestionar el acceso a la vista de roles en la aplicación. Su método index verifica si el usuario autenticado tiene el rol de administrador (role === 'admin'). Si es así, retorna la vista role con información del usuario; de lo contrario, redirige al panel de control (dashboard) con un mensaje de error indicando que no tiene acceso. Este controlador asegura que solo usuarios con permisos específicos puedan acceder a ciertas funcionalidades relacionadas con roles.



```
2 references | 0 implementations
class RoleController extends Controller
{
    1 reference | 0 overrides
    public function index(Request $request): View|RedirectResponse
    {
        if( Auth::check() && Auth::user()->role === 'admin'){
            return view(view: 'role', data: [
                'user' => $request->user(),
            ]);
        }else{
            return Redirect::route(route: 'dashboard')->with(key: 'error', value: 'You do not have access to this page.');
        }
    }
}
```

Archivo: app/Http/Controller/evento.php

Descripción: El archivo evento.php contiene la lógica para gestionar las vistas relacionadas con eventos y competencias en la aplicación. El método index busca un evento específico por su ID, obtiene sus categorías asociadas y retorna la vista competencia con estos datos. Por otro lado, el método grupos busca una competencia específica por su ID, obtiene los grupos asociados y retorna la vista grupos con esta información. Este controlador facilita la interacción entre los modelos de eventos y competencias y las vistas correspondientes, permitiendo mostrar datos dinámicos al usuario.

```php
class evento extends Controller
{
    1 reference | 0 overrides
    public function index($id_evento): View
    {
        // Busca el evento por su ID
        $evento = ModelsEvento::findOrFail(id: $id_evento);
        $categorias = $evento->categorias;

        // Retorna la vista con los datos del evento
        return view(view: 'competencia', data: compact(var_name: 'evento', var_names: 'categorias'));
    }

    1 reference | 0 overrides
    public function grupos($id_evento,$id_competencia): View
    {
        // Busca el evento por su ID
        $competencia = competencia::findOrFail(id: $id_competencia);
        $grupos = $competencia->grupos;

        // Retorna la vista con los datos del evento
        return view(view: 'grupos', data: compact(var_name: 'competencia', var_names: 'grupos'));
    }

}
```

Archivo: app/Http/Controller/CombateController.php

Descripción:

1. CreateKata

- Crea un combate de kata en la base de datos.
- Valida los parámetros de entrada y verifica que la competencia sea de kata.
- Establece el estado del combate según la fecha de inicio y guarda el ganador si se proporciona.

1.EmparejarKata

- Asigna participantes o equipos a un combate de kata.
- Verifica si el combate y los participantes existen.
- Crea equipos de kata si no existen y los asocia al combate.
- Genera presentaciones de kata para los equipos.

2.GanadorKata

- Actualiza el ganador de un combate de kata.
- Verifica que el combate exista y que sea de kata antes de guardar el ganador.

3.CrearJueces

- Crea jueces para un combate de kata.
- Verifica que el combate exista y que sea de kata.
- Asocia jueces y sus puntuaciones al combate.

4.gruposkata

- Crea un grupo de kata y lo asocia a una competencia.
- Asigna participantes o equipos al grupo y genera rondas y combates entre ellos.
- Crea jueces y emparejamientos para los combates generados.

5.CreateKumite

- Crea un combate de kumite en la base de datos.
- Valida los parámetros de entrada y verifica que la competencia sea de kumite.
- Establece el estado del combate según la fecha de inicio y guarda el ganador si se proporciona.

6.GanadorKumite

- Actualiza el ganador de un combate de kumite.
- Verifica que el combate exista y que sea de kumite antes de guardar el ganador.

7.gruposkumite

- Crea un grupo de kumite y lo asocia a una competencia.
- Asigna participantes al grupo y genera rondas y combates entre ellos.
- Asocia participantes y puntuaciones a los combates generados.

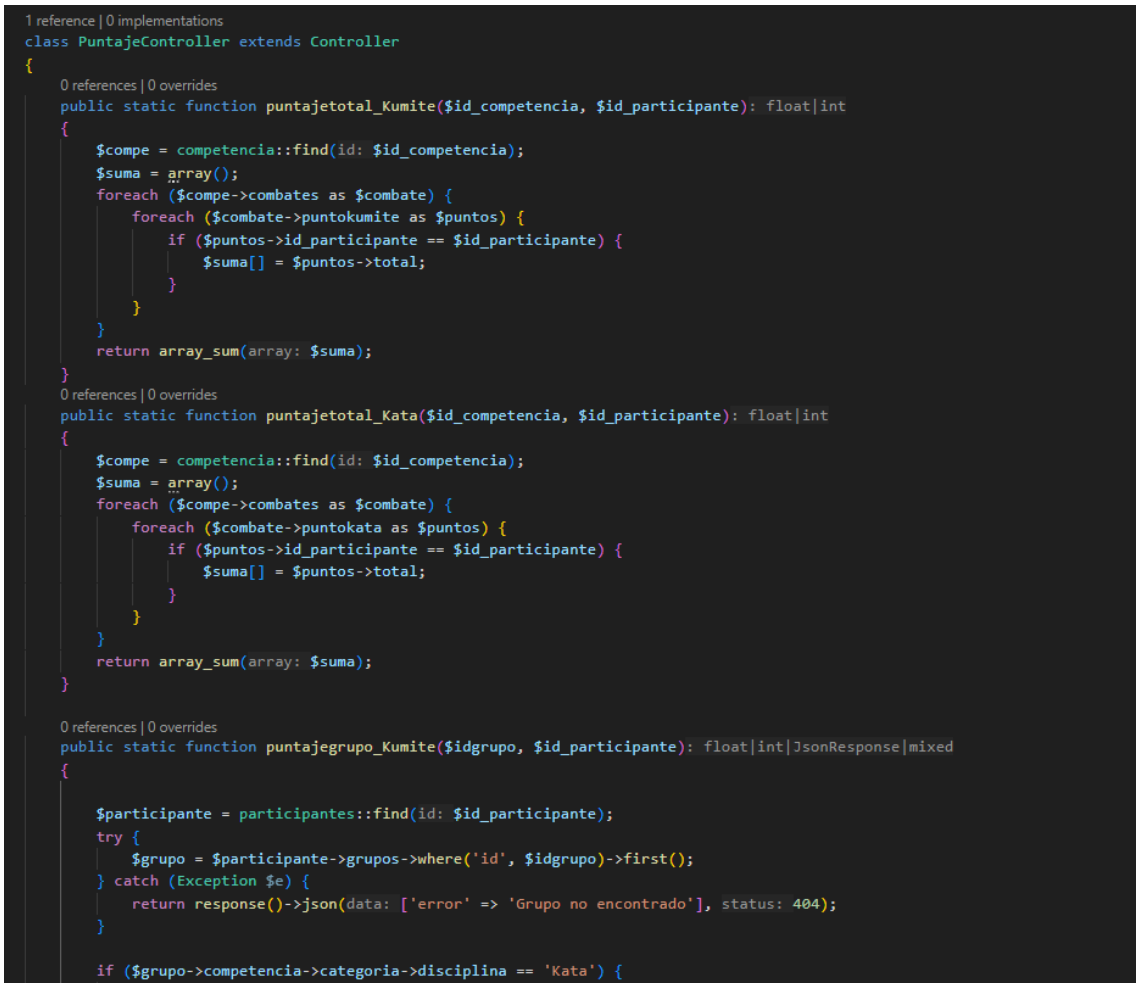
8.sortear

- Divide a 32 participantes en 8 grupos de manera aleatoria.
- Verifica que el número de participantes sea exactamente 32 antes de realizar el sorteo.

```
class CombatesController extends Controller
{
    // Métodos para usar el sistema de kata
    1 reference | 0 overrides
    public static function CreateKata($id_tatami, $id_ronda, $id_competencia, $inicia, $ganador = null): Combate|JsonResponse|mixed ···
    }

    1 reference | 0 overrides
    public static function EmparejarKata($id_combate, $id_participante1, $id_participante2): mixed ···
    }

    0 references | 0 overrides
    public static function GanadorKata($id_combate, $id_ganador): mixed ···
    }

    1 reference | 0 overrides
    public static function CrearJueces($id_combate, $num_jueces = 7): mixed ···
    }


    1 reference | 0 overrides
    public static function gruposkata($nombre, $numero, $participantes, $id_competencia, $id_tatami,$fecha=null): Collection ···
    }

    // Métodos para usar el sistema de kumite


    1 reference | 0 overrides
    public static function CreateKumite($id_tatami, $id_ronda, $id_competencia, $inicia, $ganador = null): Combate|JsonResponse|mixed ···
    }

    0 references | 0 overrides
    public static function GanadorKumite($id_combate, $id_ganador): mixed ···
    }

    1 reference | 0 overrides
    public static function gruposkumite($nombre, $numero, $participantes, $id_competencia, $id_tatami,$fecha=null): Collection ···
    }

    1 reference | 0 overrides
    public static function  sortear($allParticipants): array|JsonResponse|mixed ···
    }
}
```

Archivo: app/Http/Controller/PuntajeController.php

Descripción: El archivo PuntajeController.php contiene métodos para calcular los puntajes totales y grupales de los participantes en competencias de kumite y kata. Incluye funciones como puntajetotal_Kumite y puntajetotal_Kata, que suman los puntos obtenidos por un participante en todos los combates de una competencia, y puntajegrupo_Kumite y puntajegrupo_Kata, que calculan los puntos acumulados por un participante dentro de un grupo específico. Además, valida que los datos sean correctos y que las disciplinas coincidan con el tipo de competencia, retornando errores en caso de inconsistencias.



```
1 reference | 0 implementations
class PuntajeController extends Controller
{
    0 references | 0 overrides
    public static function puntajetotal_Kumite($id_competencia, $id_participante): float|int
    {
        $compe = competencia::find(id: $id_competencia);
        $suma = array();
        foreach ($compe->combates as $combate) {
            foreach ($combate->puntokumite as $puntos) {
                if ($puntos->id_participante == $id_participante) {
                    $suma[] = $puntos->total;
                }
            }
        }
        return array_sum(array: $suma);
    }
    0 references | 0 overrides
    public static function puntajetotal_Kata($id_competencia, $id_participante): float|int
    {
        $compe = competencia::find(id: $id_competencia);
        $suma = array();
        foreach ($compe->combates as $combate) {
            foreach ($combate->puntokata as $puntos) {
                if ($puntos->id_participante == $id_participante) {
                    $suma[] = $puntos->total;
                }
            }
        }
        return array_sum(array: $suma);
    }

    0 references | 0 overrides
    public static function puntajegrupo_Kumite($idgrupo, $id_participante): float|int|JsonResponse|mixed
    {

        $participante = participantes::find(id: $id_participante);
        try {
            $grupo = $participante->grupos->where('id', $idgrupo)->first();
        } catch (Exception $e) {
            return response()->json(data: ['error' => 'Grupo no encontrado'], status: 404);
        }

        if ($grupo->competencia->categoria->disciplina == 'Kata') {
```

Archivo: app/Http/Controller/kumite/panelcontrol.php

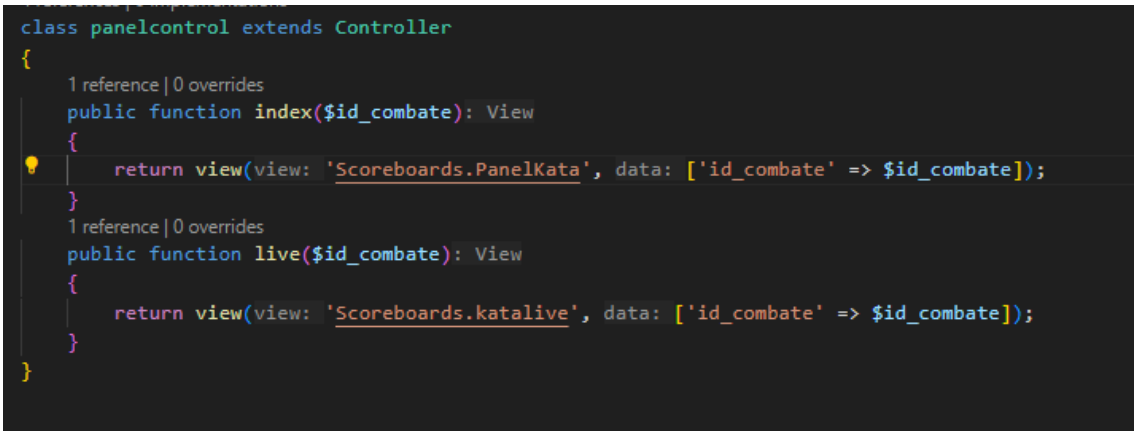
Descripción: El archivo panelcontrol.php (similar al de kata) probablemente gestiona las vistas relacionadas con el panel de control y la visualización en vivo de los

combates de kumite. Aunque no se muestra el contenido exacto, su función sería retornar vistas específicas, como un tablero de puntuación (Scoreboards.PanelKumite) y una vista en vivo (Scoreboards.kumitelive), pasando el identificador del combate como parámetro. Esto permite mostrar información dinámica sobre los combates de kumite en tiempo real o en un panel administrativo.

```php
class panelcontrol extends Controller
{
    1 reference | 0 overrides
    public function index($id_combate): View
    {
        $combate=Combate::find(id: $id_combate);
        $remaining = $combate->competencia->categoria->duracion*60;
        return view(view: 'Scoreboards.PanelKumite', data: ['id_combate'=>$id_combate,'remaining'=>$remaining]);
    }

    1 reference | 0 overrides
    public function live($id_combate): View
    {
        $combate=Combate::find(id: $id_combate);
        return view(view: 'Scoreboards.kumitelive', data: ['id_combate' => $id_combate]);
    }
}
```

Archivo: app/Http/Controller/kata/panelcontrol.php

Descripción: El archivo panelcontrol.php gestiona las vistas relacionadas con el panel de control y la visualización en vivo de los combates de kata. Contiene dos métodos: index, que retorna la vista del panel de control (Scoreboards.PanelKata) con el identificador del combate, y live, que muestra la vista en vivo del combate (Scoreboards.katalive) con el mismo identificador. Este controlador permite mostrar información específica y dinámica sobre los combates de kata en diferentes contextos, como administración o transmisión en tiempo real.



```php
class panelcontrol extends Controller
{
    1 reference | 0 overrides
    public function index($id_combate): View
    {
        return view(view: 'Scoreboards.PanelKata', data: ['id_combate' => $id_combate]);
    }
    1 reference | 0 overrides
    public function live($id_combate): View
    {
        return view(view: 'Scoreboards.katalive', data: ['id_combate' => $id_combate]);
    }
}
```

Carpeta: app/Http/Events

Descripción: Permite la transmisión en tiempo real de datos relacionados con combates de kata, kumite y el temporizador, utilizando canales de broadcasting personalizados según el identificador del combate. Esto facilita la sincronización dinámica de información entre el servidor y los clientes.

Estructura General de cada archivo:

1. **Propiedades**:
   - data: Contiene los datos que se transmitirán.
   - id_combate: Identificador del combate para personalizar el canal.
2. **Métodos**:
   - broadcastOn(): Transmite el evento en un canal público nombrado channel.{id_combate}.
   - broadcastAs(): Define el nombre del evento.
   - broadcastWith(): Envía los datos asociados al evento.

Archivo:
app/Livewire/Combatientes.php

```php
9 references | 0 implementations
class LiveKumite implements ShouldBroadcastNow
{
    use Dispatchable, InteractsWithSockets, SerializesModels;

    /**
     * Create a new event instance.
     */
    3 references
    public $data;
    2 references
    public $id_combate;
    0 references | 0 overrides
    public function __construct($data)
    {
        $this->data = $data;
        $this->id_combate = $data['id_combate'];
    }

    /**
     * Get the channels the event should broadcast on.
     *
     * @return \Illuminate\Broadcasting\Channel
     */
    0 references | 0 overrides
    public function broadcastOn(): array
    {
        return [
            new Channel(name: 'channel-Kumite.'. $this->id_combate), // Nombre del canal
        ];
    }
    /**
     * Datos que se enviarán con el evento.
     *
     * @return array
     */
    0 references | 0 overrides
    public function broadcastAs():string
    {
        return "livekumite"; // Nombre del evento que se enviará al canal
    }
    // Puedes personalizar los datos que se envían con el evento aquí
    // Por ejemplo, puedes enviar la variable $data que has pasado al constructor
    0 references | 0 overrides
    public function broadcastWith(): array
    {
        return [
            'data' => $this->data, // Enviar la variable como parte de los datos
        ];
    }
```

```php
class Combatientes extends Component
{
    public $categorias;

    public $evento;

    public $competencia;
    public $categoriaSeleccionada = null;

    public $mostrarModal = false;

    public $competenciasDisponibles = [];
    public $participantesDisponibles = [];
    public $competenciaSeleccionada;
    public $participantesSeleccionados = [];

    public $table=False;
    public $participantes;
    public function render()
    {
        return view('livewire.combatientes');
    }

    public function mount($categorias)
    {

        $this->participantesDisponibles = participantes::all();
    }

    public function grupos(){
        return redirect()->route('eventos.grupos',[$this->evento->id,$this->competencia->id]);
    }

    public function abrirModal()
    {

        $this->mostrarModal = true;
    }

    public function cerrarModal()
    {
        $this->mostrarModal = false;
    }
```

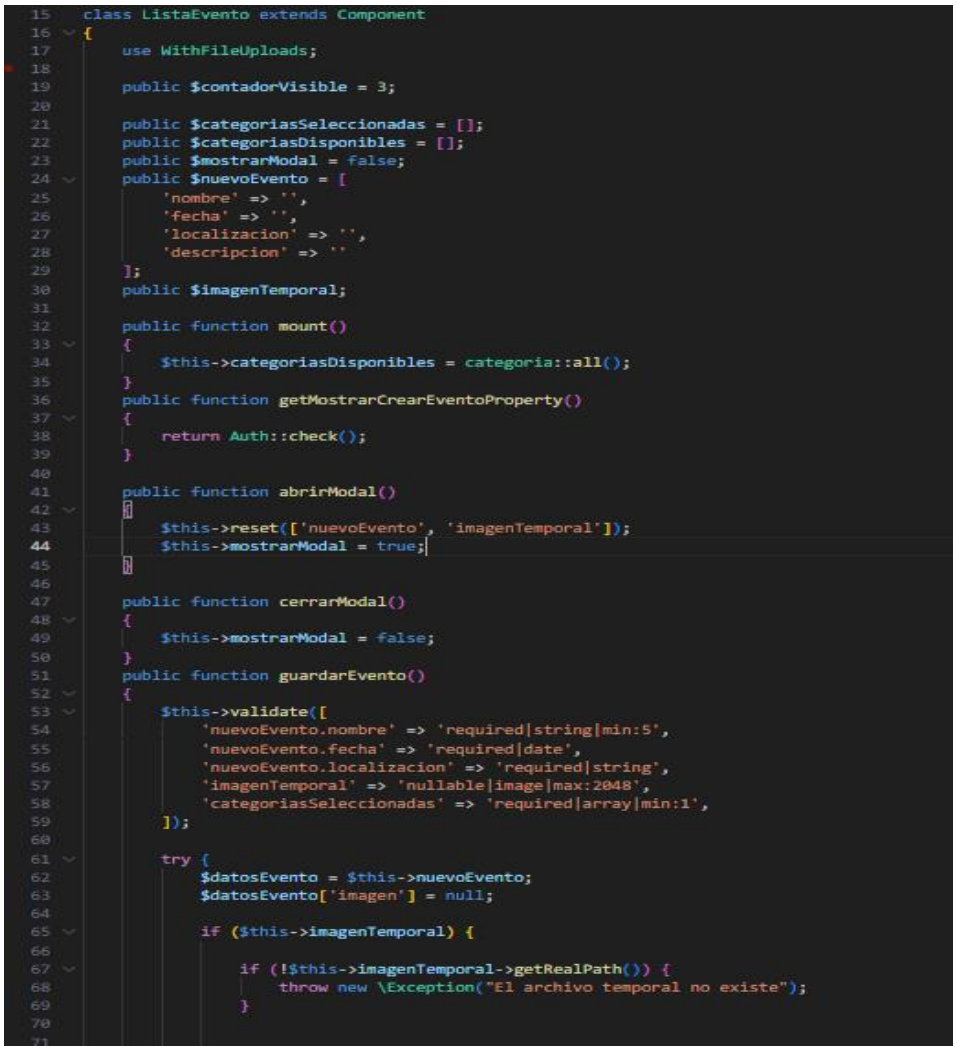
Descriocion: Gestiona participantes en una competencia. **Renderizado (**render**)** devuelve la vista livewire.combatientes, **Montaje (**mount**)** inicializa la lista de participantes disponibles, **Gestión de Modal (**abrirModal**,** cerrarModal**)** controla la visibilidad de un modal, **Guardar relación (**guardarRelacion**)** asocia participantes con una competencia después de validar los datos, **Selección automática (**seleccionar32**)** escoge los primeros 32 participantes disponibles, **Mostrar tabla (**vertabla**)** obtiene y muestra los participantes de una competencia, **Sorteo (**sorteo**)** agrupa aleatoriamente a los participantes y los asigna a diferentes tipos de combates según la disciplina (**Kumite** o **Kata**), **Redirección (**grupos**)** dirige al usuario a la vista de grupos del evento. En resumen, esta clase facilita la administración, asignación y organización de los participantes dentro de una competencia.

Archivo: app/Livewire/Grupos.php

```php
class Grupos extends Component
{
    public $grupos;
    public $competencia;

    public $grupoSeleccionado;

    public $rondaSeleccionado;

    public $combateSeleccionado;

    public $combates;

    public $rondas;

    public function ronda(){
        if($this->grupoSeleccionado!=null && $this->grupoSeleccionado!=''){
            $ronda=Ronda::where('id_grupo',$this->grupoSeleccionado)->get();
            $this->rondas=$ronda;
        }

    }
    public function combate(){
        if($this->rondaSeleccionado!=null && $this->rondaSeleccionado!=''){
            $c=Combate::where('id_ronda',$this->rondaSeleccionado)->get();
            $this->combates=$c;
        }

    }

    public function ir(){
        if($this->competencia->categoria->disciplina=='Kumite'){
            return redirect()->route('kumite.scoreboard',$this->combateSeleccionado);
        }else{
            return redirect()->route('kata.scoreboard',$this->combateSeleccionado);
        }


    }
    public function render()
    {
        return view('livewire.grupos');
    }
}
```

Descripción: Guarda información sobre grupos, rondas y combates y permite obtener las rondas asociadas a un grupo (ronda) y los combates dentro de una ronda (combate). También decide a qué tipo de marcador redirigir (ir), dependiendo de si la competencia es Kumite o Kata. Finalmente, render carga la vista livewire.grupos. En pocas palabras, organiza y facilita la navegación entre grupos, rondas y combates dentro de una competencia.

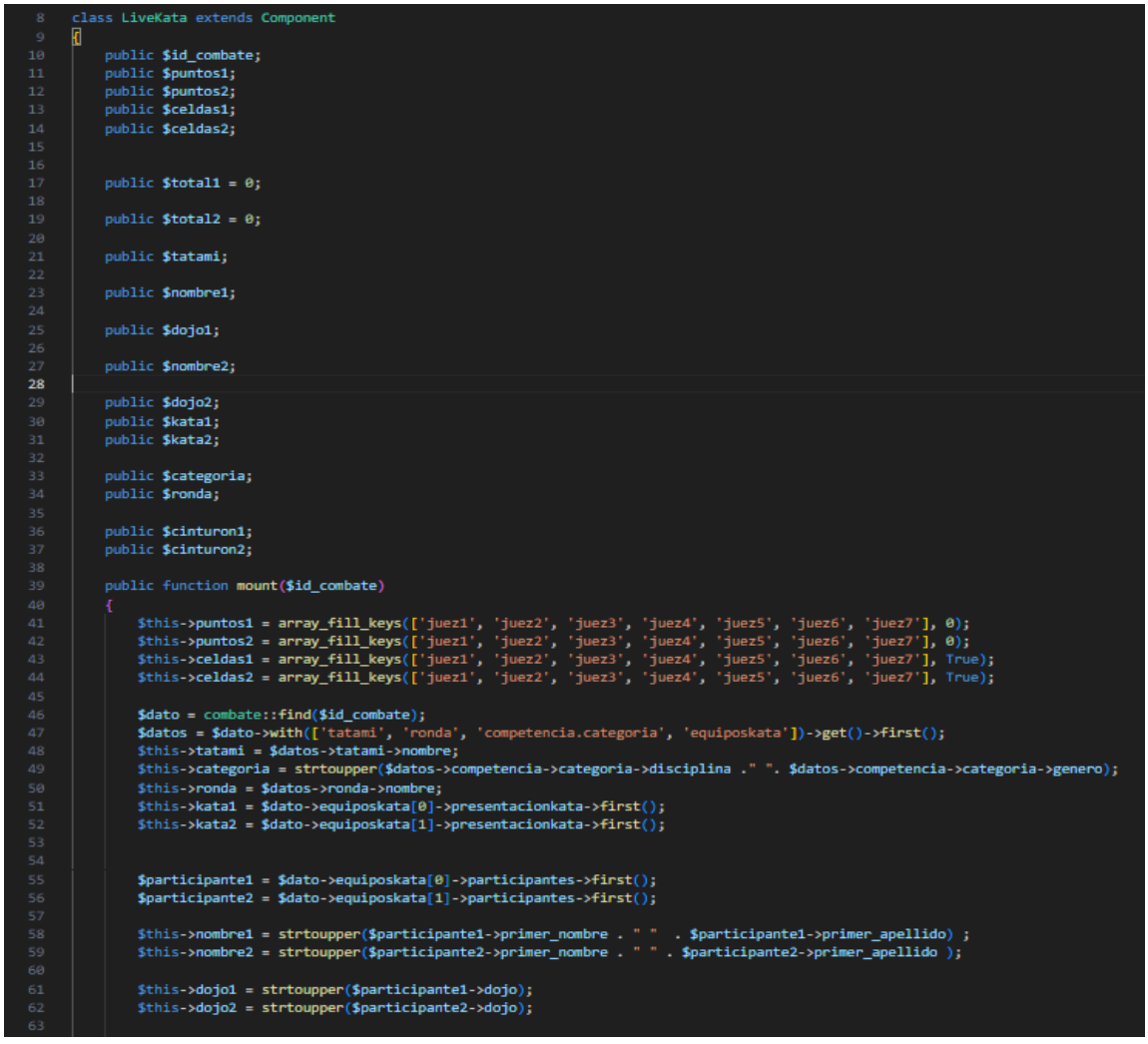
Archivo: app/Livewire/ ListaEvento.php



```php
class ListaEvento extends Component
{
    use WithFileUploads;

    public $contadorVisible = 3;

    public $categoriasSeleccionadas = [];
    public $categoriasDisponibles = [];
    public $mostrarModal = false;
    public $nuevoEvento = [
        'nombre' => '',
        'fecha' => '',
        'localizacion' => '',
        'descripcion' => ''
    ];
    public $imagenTemporal;

    public function mount()
    {
        $this->categoriasDisponibles = categoria::all();
    }
    public function getMostrarCrearEventoProperty()
    {
        return Auth::check();
    }

    public function abrirModal()
    {
        $this->reset(['nuevoEvento', 'imagenTemporal']);
        $this->mostrarModal = true;
    }

    public function cerrarModal()
    {
        $this->mostrarModal = false;
    }
    public function guardarEvento()
    {
        $this->validate([
            'nuevoEvento.nombre' => 'required|string|min:5',
            'nuevoEvento.fecha' => 'required|date',
            'nuevoEvento.localizacion' => 'required|string',
            'imagenTemporal' => 'nullable|image|max:2048',
            'categoriasSeleccionadas' => 'required|array|min:1',
        ]);

        try {
            $datosEvento = $this->nuevoEvento;
            $datosEvento['imagen'] = null;

            if ($this->imagenTemporal) {

                if (!$this->imagenTemporal->getRealPath()) {
                    throw new \Exception("El archivo temporal no existe");
                }
```

Descripción: Permite seleccionar categorías, abrir y cerrar un modal para crear eventos (abrirModal, cerrarModal), validar y guardar un nuevo evento con imagen (guardarEvento), y cargar más eventos para visualizar (cargarMasEventos). También controla cuántos eventos se muestran y verifica si hay más disponibles. Finalmente, render carga la vista livewire.lista-evento, mostrando los eventos disponibles y controlando la opción de creación. En resumen, facilita la gestión de eventos dentro de la aplicación.

Archivo: app/Livewire/ LiveKata.php



```php
class LiveKata extends Component
{
    public $id_combate;
    public $puntos1;
    public $puntos2;
    public $celdas1;
    public $celdas2;


    public $total1 = 0;

    public $total2 = 0;

    public $tatami;

    public $nombre1;

    public $dojo1;

    public $nombre2;

    public $dojo2;
    public $kata1;
    public $kata2;

    public $categoria;
    public $ronda;

    public $cinturon1;
    public $cinturon2;

    public function mount($id_combate)
    {
        $this->puntos1 = array_fill_keys(['juez1', 'juez2', 'juez3', 'juez4', 'juez5', 'juez6', 'juez7'], 0);
        $this->puntos2 = array_fill_keys(['juez1', 'juez2', 'juez3', 'juez4', 'juez5', 'juez6', 'juez7'], 0);
        $this->celdas1 = array_fill_keys(['juez1', 'juez2', 'juez3', 'juez4', 'juez5', 'juez6', 'juez7'], True);
        $this->celdas2 = array_fill_keys(['juez1', 'juez2', 'juez3', 'juez4', 'juez5', 'juez6', 'juez7'], True);

        $dato = combate::find($id_combate);
        $datos = $dato->with(['tatami', 'ronda', 'competencia.categoria', 'equiposkata'])->get()->first();
        $this->tatami = $datos->tatami->nombre;
        $this->categoria = strtoupper($datos->competencia->categoria->disciplina ." ". $datos->competencia->categoria->genero);
        $this->ronda = $datos->ronda->nombre;
        $this->kata1 = $dato->equiposkata[0]->presentacionkata->first();
        $this->kata2 = $dato->equiposkata[1]->presentacionkata->first();


        $participante1 = $dato->equiposkata[0]->participantes->first();
        $participante2 = $dato->equiposkata[1]->participantes->first();

        $this->nombre1 = strtoupper($participante1->primer_nombre . " "  . $participante1->primer_apellido) ;
        $this->nombre2 = strtoupper($participante2->primer_nombre . " " . $participante2->primer_apellido );

        $this->dojo1 = strtoupper($participante1->dojo);
        $this->dojo2 = strtoupper($participante2->dojo);
```

Descripción: Almacena información sobre los participantes, sus nombres, dojos, cinturones, categoría, ronda y puntuaciones de los jueces. En mount, se inicializan los datos del combate a partir de su ID, incluyendo los competidores y sus respectivas presentaciones de Kata. La función refresh está definida pero vacía, y render carga la vista livewire.kata.livekata2. En resumen, esta clase permite mostrar la información relevante de un combate de Kata en tiempo real.

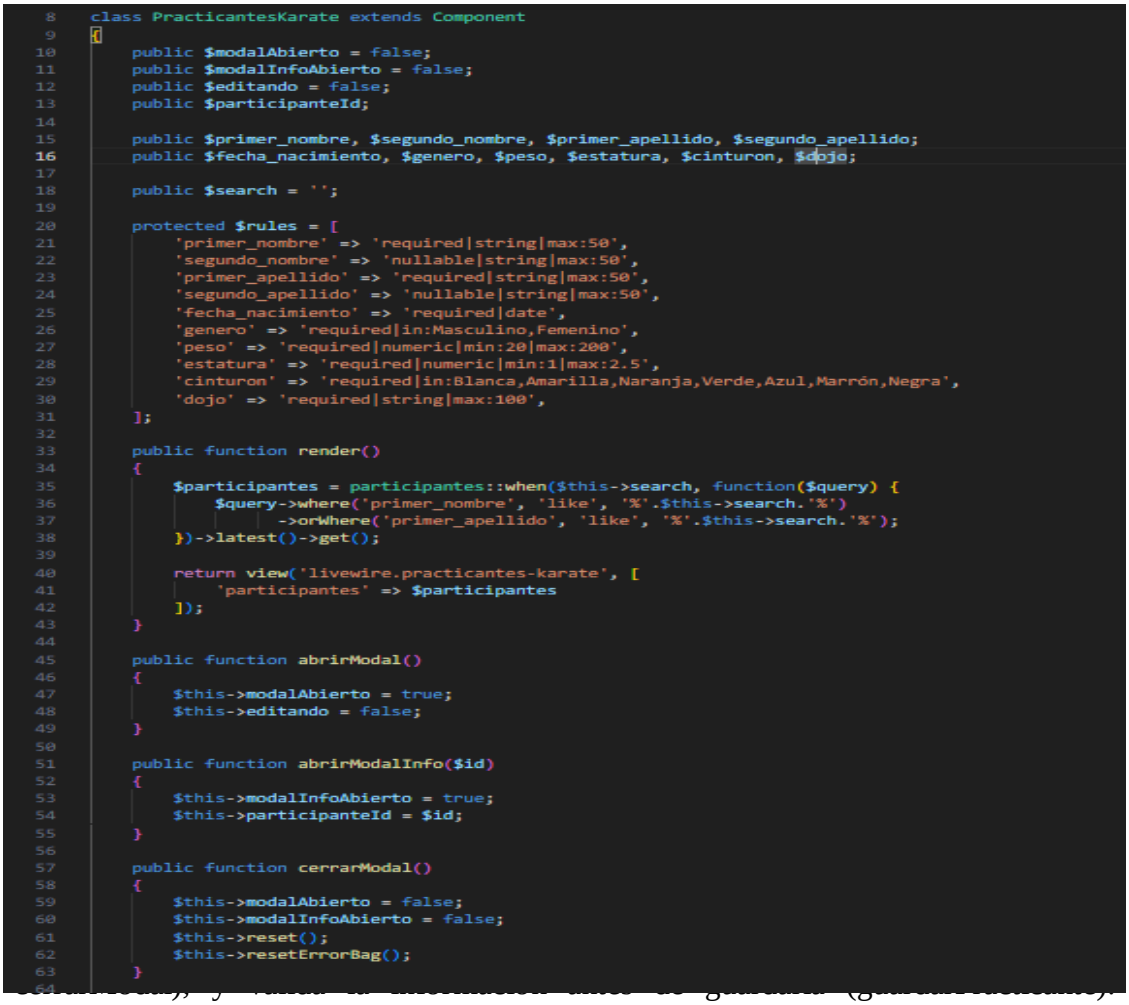
Archivo: app/Livewire/ PaneldeControlKata.php

```php
class PaneldeControlKata extends Component
{
    public $id_combate;
    public $puntos1;
    public $puntos2;
    public $celdas1;

    public $celdas2;


    public $total1 = 0;

    public $total2 = 0;

    public $tatami;

    public $nombre1;

    public $nombre2;

    public $kata1;
    public $kata2;

    public $categoria;

    public $win;

    public $participante1,$participante2;

    public function mount($id_combate)
    {

        $this->puntos1 = array_fill_keys(['juez1', 'juez2', 'juez3', 'juez4', 'juez5', 'juez6', 'juez7'], 0);
        $this->puntos2 = array_fill_keys(['juez1', 'juez2', 'juez3', 'juez4', 'juez5', 'juez6', 'juez7'], 0);
        $this->celdas1 = array_fill_keys(['juez1', 'juez2', 'juez3', 'juez4', 'juez5', 'juez6', 'juez7'], True);
        $this->celdas2 = array_fill_keys(['juez1', 'juez2', 'juez3', 'juez4', 'juez5', 'juez6', 'juez7'], True);

        $dato = combate::find($id_combate);
        $datos = $dato->with(['tatami', 'ronda', 'competencia.categoria', 'equiposkata'])->get()->first();
        $this->tatami = $datos->tatami->nombre;
        $this->categoria = $datos->competencia->categoria->disciplina . $datos->competencia->categoria->genero . "|" . $datos->ronda->nombre;

        $this->kata1 = $dato->equiposkata[0]->presentacionkata->first();
        $this->kata2 = $dato->equiposkata[1]->presentacionkata->first();

        $this->participante1 = $dato->equiposkata[0]->participantes->first();
        $this->participante2 = $dato->equiposkata[1]->participantes->first();

        $this->nombre1 = $this->participante1->primer_nombre . " "  . $this->participante1->primer_apellido . " (" . $this->participante1->dojo . ")";
        $this->nombre2 = $this->participante2->primer_nombre . " " . $this->participante2->primer_apellido . " (" . $this->participante2->dojo . ")";

        $puntoskata1 = $dato->equiposkata[0]->puntokata->first();
        $puntoskata2 = $dato->equiposkata[1]->puntokata->first();

```

Descripción: Inicializa datos del combate en mount, incluyendo participantes, nombres, dojos, tatami, categoría y puntuaciones de los jueces. La función save calcula los puntajes, descarta los valores extremos, actualiza la información en la base de datos y determina el ganador. Finalmente, render carga la vista

livewire.kata.panelkata. En resumen, organiza, evalúa y almacena los resultados de un combate de Kata.

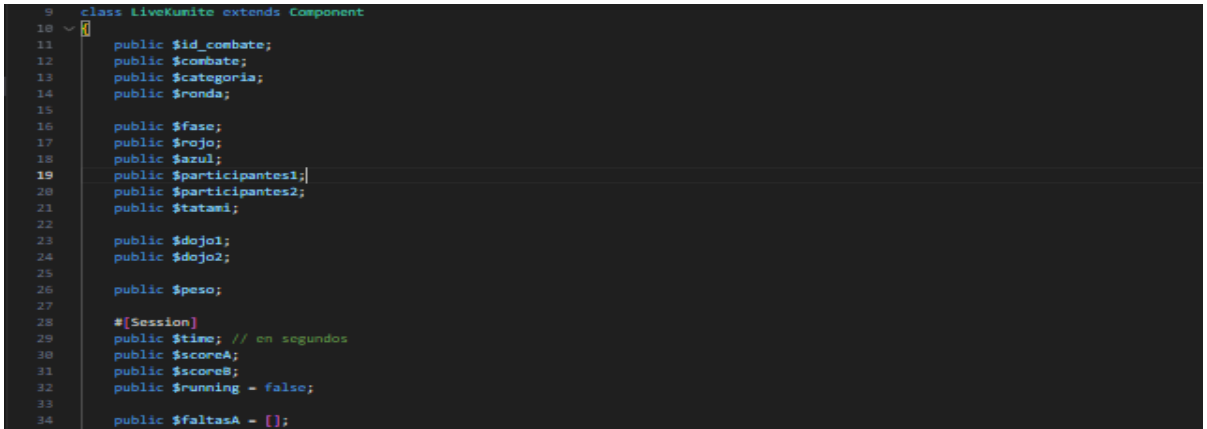
Archivo: app/Livewire/ PracticantesKarate.php



```php
class PracticantesKarate extends Component
{
    public $modalAbierto = false;
    public $modalInfoAbierto = false;
    public $editando = false;
    public $participanteId;

    public $primer_nombre, $segundo_nombre, $primer_apellido, $segundo_apellido;
    public $fecha_nacimiento, $genero, $peso, $estatura, $cinturon, $dojo;

    public $search = '';

    protected $rules = [
        'primer_nombre' => 'required|string|max:50',
        'segundo_nombre' => 'nullable|string|max:50',
        'primer_apellido' => 'required|string|max:50',
        'segundo_apellido' => 'nullable|string|max:50',
        'fecha_nacimiento' => 'required|date',
        'genero' => 'required|in:Masculino,Femenino',
        'peso' => 'required|numeric|min:20|max:200',
        'estatura' => 'required|numeric|min:1|max:2.5',
        'cinturon' => 'required|in:Blanca,Amarilla,Naranja,Verde,Azul,Marrón,Negra',
        'dojo' => 'required|string|max:100',
    ];

    public function render()
    {
        $participantes = participantes::when($this->search, function($query) {
            $query->where('primer_nombre', 'like', '%'.$this->search.'%')
                  ->orWhere('primer_apellido', 'like', '%'.$this->search.'%');
        })->latest()->get();

        return view('livewire.practicantes-karate', [
            'participantes' => $participantes
        ]);
    }

    public function abrirModal()
    {
        $this->modalAbierto = true;
        $this->editando = false;
    }

    public function abrirModalInfo($id)
    {
        $this->modalInfoAbierto = true;
        $this->participanteId = $id;
    }

    public function cerrarModal()
    {
        $this->modalAbierto = false;
        $this->modalInfoAbierto = false;
        $this->reset();
        $this->resetErrorBag();
    }
```

También permite visualizar detalles de un participante (abrirModalInfo) y asigna colores a los cinturones (getColorCinta). En resumen, facilita la gestión de practicantes dentro de la aplicación.

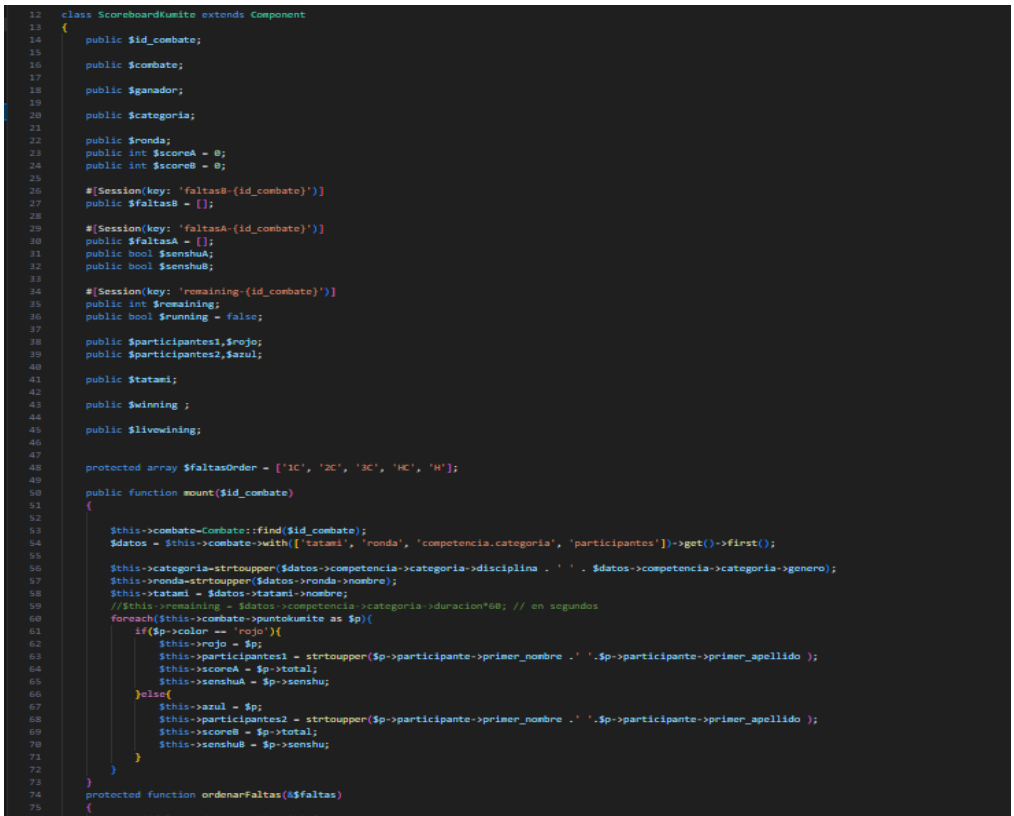
Archivo: app/Livewire/ LiveKumite.php



```php
class LiveKumite extends Component
{
    public $id_combate;
    public $combate;
    public $categoria;
    public $ronda;

    public $fase;
    public $rojo;
    public $azul;
    public $participantes1;
    public $participantes2;
    public $tatami;

    public $dojo1;
    public $dojo2;

    public $peso;

    #[Session]
    public $time; // en segundos
    public $scoreA;
    public $scoreB;
    public $running = false;

    public $faltasA = [];
```

Descripción: En mount, inicializa los datos del combate, incluyendo participantes, tatami, categoría, ronda y puntuaciones. También determina si la fase es eliminatoria o grupal. Asigna los competidores a los colores **rojo** y **azul**, y almacena información sobre faltas y ventajas **(senshu)**. Finalmente, render carga la vista livewire.kumite.livekumite2. En resumen, organiza y muestra la información clave de un combate de Kumite en tiempo real.

Archivo: app/Livewire/ ScoreboardKumite.php



```php
class ScoreboardKumite extends Component
{
    public $id_combate;

    public $combate;

    public $ganador;

    public $categoria;

    public $ronda;
    public int $scoreA = 0;
    public int $scoreB = 0;

    #[Session(key: 'faltasB-{id_combate}')]
    public $faltasB = [];

    #[Session(key: 'faltasA-{id_combate}')]
    public $faltasA = [];
    public bool $senshuA;
    public bool $senshuB;

    #[Session(key: 'remaining-{id_combate}')]
    public int $remaining;
    public bool $running = false;

    public $participantes1,$rojo;
    public $participantes2,$azul;

    public $tatami;

    public $winning ;

    public $livewining;


    protected array $faltasOrder = ['1C', '2C', '3C', 'HC', 'H'];

    public function mount($id_combate)
    {

        $this->combate=Combate::find($id_combate);
        $datos = $this->combate->with(['tatami', 'ronda', 'competencia.categoria', 'participantes'])->get()->first();

        $this->categoria=strtoupper($datos->competencia->categoria->disciplina . ' ' . $datos->competencia->categoria->genero);
        $this->ronda=strtoupper($datos->ronda->nombre);
        $this->tatami = $datos->tatami->nombre;
        //$this->remaining = $datos->competencia->categoria->duracion*60; // en segundos
        foreach($this->combate->puntokumite as $p){
            if($p->color == 'rojo'){
                $this->rojo = $p;
                $this->participantes1 = strtoupper($p->participante->primer_nombre .' '.$p->participante->primer_apellido );
                $this->scoreA = $p->total;
                $this->senshuA = $p->senshu;
            }else{
                $this->azul = $p;
                $this->participantes2 = strtoupper($p->participante->primer_nombre .' '.$p->participante->primer_apellido );
                $this->scoreB = $p->total;
                $this->senshuB = $p->senshu;
            }
        }
    }
    protected function ordenarFaltas(&$faltas)
    {
```

Descripción: Almacena información sobre los participantes, puntuaciones, faltas, tiempo restante y ganador. En mount, inicializa los datos del combate, asignando participantes a los colores rojo y azul. La función addScore incrementa la puntuación y toggleSenshu cambia la ventaja de Senshu. También maneja el temporizador (toggleTimer, resetTimer, decrementTimer) y determina el ganador en ganador. Finalmente, render actualiza la vista livewire.kumite.scoreboard-kumite. En resumen, esta clase supervisa y registra el desarrollo del combate en tiempo real.