SISTEMA DE INFORMACIÓN PARA LA GESTIÓN DE UN TORNEO DE FUTBOL EN LARAVEL

HERI FERNANDO LONDOÑO SALGADO

Instrucciones

El presente instructivo consta de tres tipos de contenido que son la teoría, las actividades y las tareas.

- **La teoría.** Contextualiza al aprendiz sobre el conocimiento que va a adquirir.
- **Las actividades.** Corresponden a las líneas de código y comandos representados en imágenes, que el aprendiz debe realizar.
- **Las tareas.** Son ejercicios que se dejan para que el aprendiz realice con base en la teoría y las actividades.

Que es Laravel

- Framework PHP de código abierto, robusto y fácil de entender.
- Sigue un patrón de diseño modelo-vista-controlador.
- Reutiliza los componentes existentes, creando aplicaciones más estructuradas y pragmáticas.
- Ofrece un conjunto de funcionalidades y características que aumentan la velocidad del desarrollo web.
- Permite crear aplicaciones seguras evitando ataques web.

Ventajas de Laravel

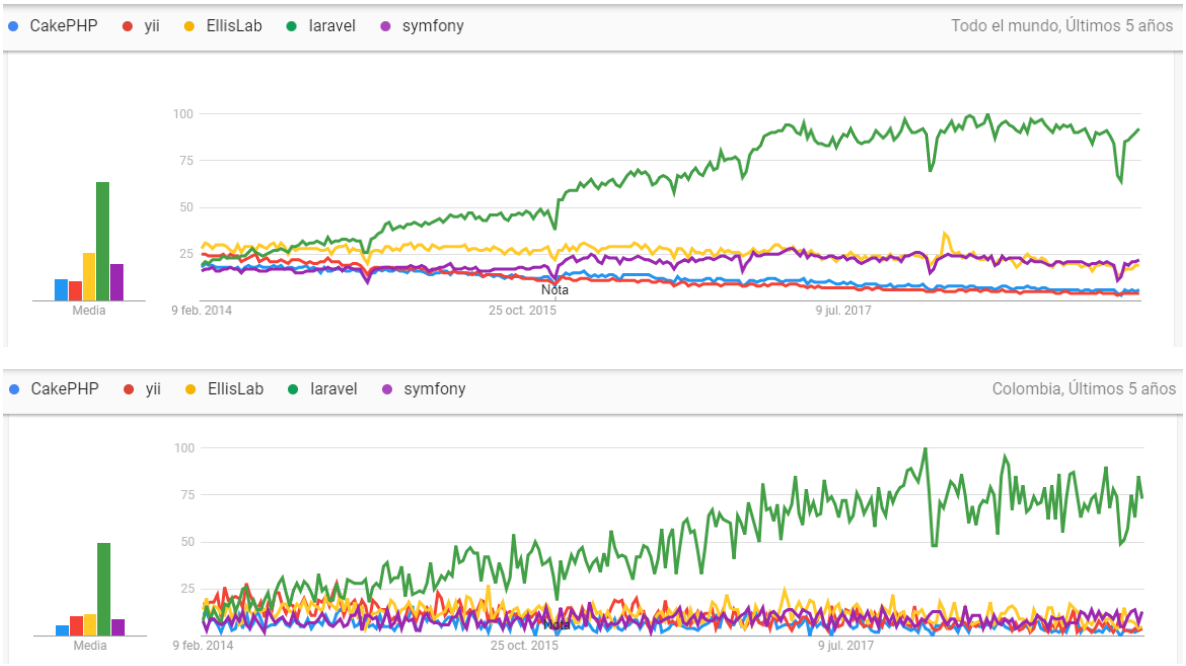
- La aplicación web es más escalable.
- Se ahorra mucho tiempo en el diseño de la aplicación por la reutilización de componentes.
- Incluye espacios de nombres e interfaces, por lo tanto, ayuda a organizar y administrar recursos.

Características de Laravel

- **Modularidad.** 20 bibliotecas y módulos integrados con el administrador de dependencias de Composer, que ayudan a mejorar la aplicación.
- **Testeabilidad.** Incluye funciones para ejecución de pruebas a través de casos de prueba.
- **Enrutamiento.** Permite al usuario definir rutas en la aplicación, permitiendo escalar la aplicación y aumenta su rendimiento.
- **Gestión de la configuración.** Una aplicación diseñada en Laravel se ejecutará en diferentes entornos, con un manejo eficiente de la configuración.
- **Generador de consultas y ORM (Object Relational Mapper).** Incorpora un generador de consultas para acceso y consulta de bases de datos utilizando varios métodos de cadena simples.

- **Constructor de esquemas.** Mantiene la definición y esquema de la base de datos en código PHP, haciendo seguimiento de los cambios con respecto a las migraciones de la base de datos.
- **Motor de plantillas.** Usa el motor de plantillas Blade, un lenguaje de plantillas ligero para la creación de contenido dinámico.
- **Email.** Incluye una clase de correo que ayuda a enviar correo con contenido enriquecido y archivos adjuntos desde la aplicación web.
- **Autenticación.** Facilita el diseño de la autenticación, ya que incluye características como el registro, la contraseña olvidada y el envío de recordatorios de contraseña.
- **Redis.** Laravel usa Redis para conectarse a una sesión existente y caché de propósito general. Redis interactúa con la sesión directamente.
- **Colas.** Laravel incluye servicios de cola, como enviar por correo electrónico a un gran número de usuarios o una tarea cron específica. Estas colas ayudan a completar las tareas de una manera más fácil sin esperar a que se complete la tarea anterior.
- **Bus de Evento y Comando.** Incluye Command Bus, que ayuda a ejecutar comandos y enviar eventos de manera sencilla. Los comandos en Laravel actúan según el ciclo de vida de la aplicación.

Tendencias entre los Frameworks de PHP



Fuente.

https://trends.google.es/trends/explore?date=today%205-y&q=%2Fm%2F09t3sp,yii,%2Fm%2F02qgdkj,laravel,symfony

Modelo Vista Controlador (MVC)

Es un estilo de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos. El estilo de llamada y retorno MVC, se ve frecuentemente en aplicaciones web, donde la vista es la página HTML y el código que provee de datos dinámicos a la página. El modelo es el Sistema de Gestión de Base de Datos y la Lógica de negocio, y el controlador es el responsable de recibir los eventos de entrada desde la vista.

- **Modelo.** Es donde esta toda la lógica del negocio, la representación de todo el sistema incluido la interacción con una base de datos, si es que el programa asi lo requiere.
- **Vista.** Representa la interfaz de usuario y todas las herramientas con las cuales el usuario hace uso del programa.
- **Controlador.** Es el que escucha los cambios en la vista y se los envía al modelo, el cual le regresa los datos a la vista, es un ciclo donde cada acción del usuario causa que se inicie de nuevo un nuevo ciclo. En cierta forma debe tener un registro de la relación entre ordenes que le pueden llegar y la lógica de negocio que le corresponde.

Como funciona MVC

1. El usuario envía una petición
2. Se redirecciona la petición al controlador Laravel.
3. El controlador interactúa con el modelo de datos.
4. Se invoca el resultado a la vista.
5. La vista se renderiza en el navegador.

Requisitos de Laravel

- PHP >= 7.3

- BCMath PHP Extension
- Ctype PHP Extension
- Fileinfo PHP Extension
- JSON PHP Extension
- Mbstring PHP Extension
- OpenSSL PHP Extension
- PDO PHP Extension
- Tokenizer PHP Extension
- XML PHP Extension

Composer

Una forma de instalar Laravel es a través de Composer para la gestión de dependencias.

Es necesario descargar e instalar Composer desde la siguiente URL.

https://getcomposer.org/download/

Después de instalar Composer, se debe abrir CMD y comprobar la instalación tecleando el comando composer. Se debe mostrar el resultado de la figura.

Actividad 1

Descargar, instalar y probar Composer.

Instalar Laravel

Se utiliza Composer para instalar Laravel de esta manera:

composer create-project laravel/laravel example-app

Con este comando se crea un nuevo proyecto llamado mi-proyecto. Composer descarga los archivos del repositorio de distribución.

Para instalar Laravel en una versión diferente a la última, se especifica de esta manera:

composer create-project laravel/laravel miproyecto "6.*"

Para ver y navegar por la aplicación recién creada se ejecuta el comando:

php artisan serve.

Este comando ejecuta la aplicación en la url http://localhost:8000. Para detener la ejecución del servidor, se presiona Ctrl + C

Actividad 2

Crear una carpeta en una ruta deseada del disco duro para alojar su proyecto.

Navegar en el CMD hasta esa ruta.

Instalar Laravel escribiendo y ejecutando el comando mostrado en la figura

```
composer create-project laravel/laravel TorneoV9
```

El proceso de descarga puede tardar unos minutos.

Para ver y navegar por la aplicación recién creada se debe ingresar a la carpeta del proyecto y ejecutar el comando:

php artisan serve

Este comando ejecuta la aplicación en la url http://localhost:8000.

Para detener la ejecución del servidor, se presiona Ctrl + C

Actividad 3

Ingresar a la carpeta del proyecto

Cargar el servicio con el comando mostrado.

```
php artisan serve
```

Cuando el servicio esta cargado mostrara la URL donde se despliega

```
Laravel development server started: <http://127.0.0.1:8000>
```

Abrir el navegador e ir a esa URL para ver el proyecto en ejecución.

Front Controller

Laravel funciona con un patrón de diseño Front Controller, este consiste en que un solo componente de la aplicación es el responsable de manejar de todas las peticiones HTTP que ésta recibe.

Todas las peticiones web entran a través del archivo public/index.php, quien comprueba en el archivo de rutas (routes/web.php) si la URL es válida y a que controlador tiene que hacer la petición.

Luego se llamará al método del controlador asignado para dicha ruta. Dependiendo de la petición, se obtienen los datos necesarios para pasarlos a la vista.

Rutas

Son una capa muy importante en Laravel, posee su propia carpeta, (routes), donde se encuentran los archivos web.php y api.php.

Las rutas de la aplicación se deben definir en web.php, cualquier ruta no definida allí no será válida, generado una excepción (devuelve un error 404).

Las rutas, en su forma más sencilla, pueden devolver directamente un valor desde el propio archivo de rutas, pero también podrán generar la llamada a una vista o a un controlador.

Al definir la ruta, se debe definir el método HTML, la URL de la petición y el llamado a una vista, a un controlador o el retorno de un dato.

```
Route::get($uri, $callback);
Route::post($uri, $callback);
Route::put($uri, $callback);
Route::patch($uri, $callback);
Route::delete($uri, $callback);
Route::options($uri, $callback);
```

```
Route::get('services', function () {
    return "Listado de servicios";
});
Route::get('services', function () {
    return view('services.index');
});
Route::get('services', 'CatalogController@index');
```

Rutas con parámetros

Para añadir parámetros a una ruta, estos se indican entre llaves {} a continuación de la ruta. Es posible definir tantos parámetros como se requieran en la ruta. También es posible indicar que un parámetro es opcional, añadiendo el símbolo “?”, y asignarle un valor por defecto.

```php
Route::get('user/{id}', function($id)
{
    return 'User '.$id;
});
Route::get('user/{name?}', function($name = null)
{
    return $name;
});
Route::get('user/{name?}', function($name = 'Javi')
{
    return $name;
});
```

Actividad 4

Escribir en el archivo de rutas, (web.php), el código mostrado en la imagen, para crear las rutas mostradas en la tabla. Probar en el navegador.

Ruta	Texto
/	Pantalla principal
equipos	Listado de equipos
equipos/create	Crear equipo
equipos/{id}	Detalle de equipo {id}
equipos/{id}/edit	Modificar equipo {id}

```php
Route::get('/', function(){
  return "Pantalla principal";
});
Route::get('equipos', function(){
  return "Listado de equipos";
});
Route::get('equipos/create', function(){
  return "Crear equipo";
});
Route::get('equipos/{id}', function($id){
  return "Detalle de equipo $id";
});
Route::get('equipos/{id}/edit', function($id){
  return "Modificar equipo $id";
});
```

Tarea

Escribir en el archivo de rutas, (web.php), el código necesario para crear las rutas mostradas en la tabla. Probar en el navegador.

Ruta	Texto
Jugadores	Listado de jugadores
jugadores/créate	Crear jugador
jugadores/{id}	Detalle de jugador {id}
jugadores/{id}/edit	Modificar jugador {id}

Vistas

La vista se retorna desde una ruta o desde un controlador con el helper view, y se envía como primer parámetro el nombre del archivo, sin extensión, y de ser necesario, como segundo parámetro un array de datos que se le pasarán a la vista.

las vistas pueden estar anidadas dentro de cualquier carpeta dentro de resources/views. En este caso se usa dot notation para hacer referencia a vistas anidadas, donde las barras que separan las carpetas se sustituyen por puntos

Pasar datos a una vista.

Para pasar datos a una vista se debe utilizar el segundo parámetro del método view, que acepta un array asociativo.

return view('greeting', ['name' => 'James']);

Laravel además ofrece una alternativa, donde en lugar de pasar un array como segundo parámetro se utiliza el método with para indicar una a una las variables o contenidos a enviar a la vista.

Dentro de la vista, se pueden acceder a estos valores utilizando su clave correspondiente.

Para imprimir una variable en la vista, se usa la sintaxis de dobles llaves {{ $var }} de Blade.

```php
Route::get('/', function () {
    return view('home');
});
Route::get('catalog/show/{id}', function ($id) {
    return view('catalog.show', array('id' => $id));
});
Route::get('catalog/show/{id}', function ($id) {
    return view('catalog.show')->with('id', $id));
});
```

Actividad 5

Crear los archivos, para las vistas, definidos en las tablas y agregar el contenido definido en cada imagen. Se debe crear una subcarpeta llamada equipos.

Archivo	Imagen	Ruta
resouses/views/inicio.blade.php	1	/
resouses/views/equipos/index.blade.php	2	equipos
resouses/views/equipos/create.blade.php	3	equipos/create
resouses/views/equipos/show.blade.php	4	equipos/{id}
resouses/views/equipos/edit.blade.php	5	equipos/{id}/edit

Imagen 1. `<h1>Pantalla principal</h1>`

Imagen 2. `<h1>Listado de equipos</h1>`

Imagen 3. `<h1>Crear equipo</h1>`

Imagen 4. `<h1>Detalle de equipo {{$id}}</h1>`

Imagen 5. `<h1>Modificar equipo {{$id}}</h1>`

Modificar en el archivo de rutas, (web.php), con el código mostrado en la imagen, para retornar las vistas dentro de las rutas. Probar en el navegador.

```php
Route::get('/', function(){
    return view('inicio');
});
Route::get('equipos', function(){
    return view('equipos.index');
});
Route::get('equipos/create', function(){
    return view('equipos.create');
});
Route::get('equipos/{id}', function($id){
    return view('equipos.show')->with('id', $id);
});
Route::get('equipos/{id}/edit', function($id){
    return view('equipos.edit')->with('id', $id);
});
```

Tarea

Crear una carpeta llamada “jugadores” dentro de “resouses/views”, y dentro de esta crear los archivos para las vistas y agregar el contenido definido en la tabla. Crear las respectivas rutas. Probar en el navegador.

Archivo	Texto	Ruta

index.blade.php	Listado de jugadores	jugadores
create.blade.php	Crear jugador	jugadores/create
show.blade.php	Detalle de jugador {id}	jugadores/{id}
edit.blade.php	Modificar jugador {id}	jugadores/{id}/edit

Plantillas Blade

Blade es un motor de plantillas incluido con Laravel. Las vistas en Blade usan la extensión blade.php y se almacenan en el directorio resources/views.

Blade permite la definición de layouts para crear una estructura HTML base con secciones que serán rellenadas por otras plantillas o vistas hijas.

Las vistas que extienden un layout deben sobrescribir las secciones del layout.

La directiva @section permite añadir contenido en las plantillas hijas, mientras que @yield será sustituido por el contenido que se indique.

El método @yield también permite establecer un contenido por defecto mediante su segundo parámetro:

@yield('section', 'Contenido por defecto')

```
<html>
  <head>
    <title>App Name - @yield('title')</title>
  </head>
  <body>
    @section('sidebar')
      This is the master sidebar.
    @show
    <div class="container">
      @yield('content')
    </div>
  </body>
</html>
```

Heredar una plantilla

Cuando se define una vista hija, se utiliza la directiva Blade @extends para especificar de qué layout debe "heredar". Las vistas que extienden un layout de Blade pueden inyectar contenido en las secciones mediante las directivas @section.

La directiva @parent será reemplazada por el contenido del layout cuando se procese la vista.

La directiva @endsection definirá únicamente una sección mientras que @show definirá y enlazará inmediatamente la sección.

```
@extends('layouts.app')
@section('title', 'Page Title')
@section('sidebar')
  @parent
  <p>This is appended to the master sidebar.</p>
@endsection
@section('content')
  <p>This is my body content.</p>
@endsection
```

This is the master sidebar.

This is appended to the master sidebar.

This is my body content.

Incluir una plantilla

En Blade se puede incluir el contenido de una plantilla dentro de otra, con la instrucción @include.

@include('view_name')

Además, se puede pasar un array de datos a la plantilla a cargar usando el segundo parámetro del método include.

@include('view_name', array('some'=>'data'))

Esta opción es muy útil para crear vistas que sean reutilizables o para separar el contenido de una vista en varios archivos.

Si se intenta @include en una vista que no existe, Laravel lanzará un error. Si desea incluir una vista que puede o no estar presente, debe utilizar la directiva @includeIf.

@includeIf('view.name', ['some' => 'data'])

Actividad 6

A continuación, se creará el layout base que utilizaran las vistas del sitio web y se incluirá la librería Bootstrap para utilizarla como estilo base.

Crear la carpeta layouts dentro de resources/views.

Dentro de esa carpeta layouts crear el archivo base.blade.php

Dentro del archivo base.blade.php se copia el contenido de la plantilla base que propone Bootstrap.

https://getbootstrap.com/docs/4.6/getting-started/introduction/#starter-template

y para incluir Blade, se modificará lo siguiente.

La etiqueta title será reemplazada por.

<title>Torneo - @yield('title')</title>

Dentro de la sección <body> se elimina el texto que viene de ejemplo y se incluye la referencia a la barra de navegación que se definirá más adelante.

@include('partials.navbar')

Luego se añade la sección donde aparecerá el contenido.

@yield('content')

Por lo tanto, la plantilla debe quedar como se muestra en la imagen.

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.6.0/dist/css/bootstrap.min.css"
    integrity="sha384-B0vP5xmATw1+K9KRQjQERJvTumQW0nPEzvF6L/Z6nronJ3oUOFUFpCjEUQouq2+l" crossorigin="anonymous
    ">
    <title>Torneo - @yield('title')</title>
  </head>
  <body>
    <div class="container">
        @include('partials.navbar')
        @yield('content')
    </div>
    <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js" integrity="sha384-DfXdz2htPH0lsSSs5nCTpuj/
    zy4C+OGpamoFVy38MVBnE+IbbVYUew+OrCXaRkfj" crossorigin="anonymous"></script>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@4.6.0/dist/js/bootstrap.bundle.min.js" integrity="
    sha384-Piv4xVNRyMGpqkS2by6br4gNJ7DXjqk09RmUpJ8jgGtD7zP9yug3goQfGII0yAns" crossorigin="anonymous"></script>
  </body>
</html>
```

Nota. En caso de no querer usar Bootstrap de manera remota, sino local, se deben descargar los archivos css y js y guardarlos respectivamente en las carpetas public/css y public/js, y referenciarlos con la función asset, con la cual se ubica la ruta relativa a partir de la carpeta public.

```
<link rel="stylesheet" href="{{ asset('css/bootstrap.min.css') }}">
```

```
<script src="{{ asset('js/jquery-3.4.1.slim.min.js') }}"></script>
```

A continuación, se creará la plantilla con el menú.

Crear la carpeta partials dentro de resources/views.

Dentro de esa carpeta partials crear el archivo navbar.blade.php

Dentro del archivo se creará el un menú basado en Bootstrap que se muestra en la imagen.

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
    <a class="navbar-brand" href="#">Torneo</a>
    <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarSupportedContent"
    aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarSupportedContent">
      <ul class="navbar-nav mr-auto">
        <li class="nav-item dropdown">
          <a class="nav-link dropdown-toggle" href="#" id="navbarDropdown" role="button" data-toggle="dropdown
          " aria-haspopup="true" aria-expanded="false">
            Equipos
          </a>
          <div class="dropdown-menu" aria-labelledby="navbarDropdown">
            <a class="dropdown-item" href="/equipos">Ver equipos</a>
            <a class="dropdown-item" href="/equipos/create">Nuevo equipo</a>
          </div>
        </li>
        <li class="nav-item dropdown">
            <a class="nav-link dropdown-toggle" href="#" id="navbarDropdown" role="button" data-toggle="
            dropdown" aria-haspopup="true" aria-expanded="false">
              Jugadores
            </a>
            <div class="dropdown-menu" aria-labelledby="navbarDropdown">
              <a class="dropdown-item" href="/jugadores">Ver jugadores</a>
              <a class="dropdown-item" href="/jugadores/create">Nuevo jugador</a>
            </div>
          </li>
      </ul>
    </div>
  </nav>
```

Modificar las vistas para heredar del layout base, reemplazando el title y mostrar el contenido, de acuerdo con las imágenes.

resouses/views/inicio.blade.php

```
@extends('layouts.base')
@section('title', 'Torneo')
@section('content')
    <h1>Pantalla principal</h1>
@endsection
```

resouses/views/equipos/index.blade.php

```
@extends('layouts.base')
@section('title', 'Equipos')
@section('content')
    <h1>Listado de equipos</h1>
@endsection
```

resouses/views/equipos/create.blade.php

```
@extends('layouts.base')
@section('title', 'Crear equipo')
@section('content')
    <h1>Crear equipo</h1>
@endsection
```

resouses/views/equipos/show.blade.php

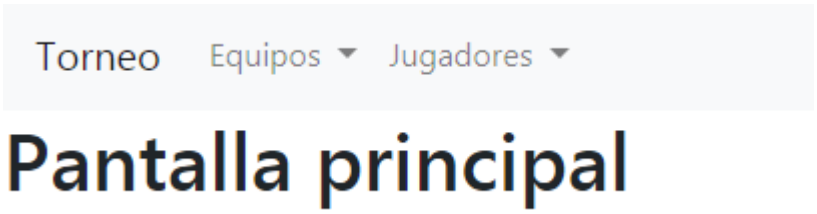
```
@extends('layouts.base')
@section('title', 'Detalle equipo')
@section('content')
    <h1>Detalle de equipo {{ $id }}</h1>
@endsection
```

resouses/views/equipos/edit.blade.php

```
@extends('layouts.base')
@section('title', 'Modificar equipo')
@section('content')
    <h1>Modificar equipo {{ $id }}</h1>
@endsection
```

Probar en el navegador.

La imagen muestra el resultado para la pantalla principal.



Tarea

Crear las vistas de los jugadores, (index, create, show y edit), de tal forma que hereden del layout base.

Probar en el navegador.

Controladores

Un controlador es una clase que permite agrupar las acciones de una sección en particular. Normalmente se les añade el sufijo Controller, por ejemplo, ProductosController, ContactosController, RegistrosController. Y los métodos de ProductosController serían, por ejemplo, “crear”, “modificar”, “eliminar”, “listar”, etc.

Los controladores se almacenan en el directorio app/Http/Controllers.

Cada una de sus acciones se definen dentro de routes/web.php.

```
use App\Http\Controllers\MiControladorController;
Route::get('/user/{id}', [MiControladorController::class, 'metodo']);
```

Para crear un controlador se puede utilizar el comando de Artisan.

php artisan make:controller NombreControladorController

Este comando creará el controlador dentro de la carpeta app/Http/Controllers y lo completará con el código básico.

Actividad 7

Crear el controlador InicioController, ejecutando el comando mostrado.

```
>php artisan make:controller InicioController
```

Nota. Si el servidor está en ejecución, presione Ctrl + C

El archivo creado tendrá el contenido mostrado en la imagen.

```php
<?php
namespace App\Http\Controllers;
use Illuminate\Http\Request;
class InicioController extends Controller
{
}
```

Al controlador creado se le agregará el método index, el cual retornará la vista relacionada

```php
<?php
namespace App\Http\Controllers;
use Illuminate\Http\Request;
class InicioController extends Controller
{
    public function index(){
        return view('inicio');
    }
}
```

Dentro del archivo de rutas, modificar la ruta '/' de tal forma que haga referencia al controlador y al método.

```php
use App\Http\Controllers\InicioController;
```

```php
Route::get('/', [InicioController::class, 'index']);
```

Resource Controller

Artisan permite crear un controlador que realiza todas las acciones del CRUD, y además de eso enrutarlas con una línea de código.

Para crear el controlador se usa el comando.

php artisan make:controller NombreControladorController --resource

Este comando crea un controlador con los siguientes métodos

index: Mostrar registros existentes (GET)

create: Mostrar nuevos datos a crear (GET)

store: Crear nuevos datos (POST)

show: Mostrar un registro (GET)

edit: Mostrar un registro a modificar (GET)

update: Modificar un registro (PUT)

destroy: Eliminar un registro (DELETE)

En la tabla se presenta el verbo, la URI, y la acción por defecto generadas por el Resource Controller, y la forma como se manejan las rutas.

Verbo	URI	Acción	Vista
GET	equipos	index	equipos.index
GET	equipos/create	create	equipos.create
POST	equipos	store	equipos.store
GET	equipos/{id}	show	equipos.show
GET	equipos/{id}/edit	edit	equipos.edit
PUT	equipos/{id}	update	equipos.update
DELETE	equipos/{id}	destroy	equipos.destroy

Para que este controlador pueda ser accedido, se debe hacer su llamado dentro del archivo de rutas, routes/web.php.

Route::resource(‘ruta’, NombreControladorController::class);

Es posible registrar varios controladores recurso pasando un arreglo al método resources.

```
Route::resources([
    'photos' => PhotoController::class,
    'posts' => PostController::class,
```

]);

Actividad 8

A continuación, se creará el Resource Controller EquiposController, este generará automáticamente las acciones para el CRUD.

En la consola escribir el comando mostrado en la imagen.

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

El archivo creado, app/Http/Controllers/EquiposController, posee 7 métodos relacionados con el CRUD. A continuación, se agregará el contenido a los métodos index, create, show, y edit, como se muestra en la imagen, para el cargue de vistas.

```php
<?php
namespace App\Http\Controllers;
use Illuminate\Http\Request;
class EquiposController extends Controller
{
    public function index()
    {
        return view('equipos.index');
    }
    public function create()
    {
        return view('equipos.create');
    }
    public function store(Request $request)
    {
    }
    public function show($id)
    {
        return view('equipos.show')
                ->with('id', $id);
    }
    public function edit($id)
    {
        return view('equipos.edit')
                ->with('id', $id);
    }
    public function update(Request $request, $id)
    {
    }
    public function destroy($id)
    {
    }
}
```

A continuación, en el archivo de rutas se define la ruta del recurso, y se pueden eliminar las rutas individuales hacia las acciones de equipos, por lo tanto, el archivo de rutas quedará como se muestra en la imagen.

```php
<?php
use Illuminate\Support\Facades\Route;
use App\Http\Controllers\InicioController;
use App\Http\Controllers\EquiposController;

Route::get('/', [InicioController::class, 'index']);
Route::resource('equipos', EquiposController::class);
```

Probar en el navegador.

Tarea

crear el Resource Controller JugadoresController,

agregar el contenido a los métodos index, create, show, y edit, para el cargue de vistas.

definir la ruta del recurso.

Actividad 9

Con el fin de simular un comportamiento dinámico, se agregará un array de equipos con el nombre, el dt, el municipio y el escudo, en el controlador EquiposController.

Agregar el array mostrado como atributo de EquiposController

```php
private $equipos = array(
    array(
        'nombre' => 'equipo 1',
        'dt' => 'D.T. 1',
        'municipio' => 'Municipio 1',
        'escudo' => 'http://ximg.es/200x300/000/fff&text=equipo1'
    ),
    array(
        'nombre' => 'equipo 2',
        'dt' => 'D.T. 2',
        'municipio' => 'Municipio 2',
        'escudo' => 'http://ximg.es/200x300/000/fff&text=equipo2'
    ),
    array(
        'nombre' => 'equipo 3',
        'dt' => 'D.T. 3',
        'municipio' => 'Municipio 3',
        'escudo' => 'http://ximg.es/200x300/000/fff&text=equipo3'
    ),
    array(
        'nombre' => 'equipo 4',
        'dt' => 'D.T. 4',
        'municipio' => 'Municipio 4',
        'escudo' => 'http://ximg.es/200x300/000/fff&text=equipo4'
    )
);
```

El index de equipos mostrará el listado de los equipos inscritos en el torneo. Los equipos están disponibles en el array creado como atributo, (más adelante se importarán desde la base de datos).

En el método index del controlador se modifica la generación de la vista para pasarle el array de equipos.

```php
public function index()
{
  return view('equipos.index')
              ->with('equipos', $this->equipos);
}
```

En la vista index se agrega el código html en la sección content para que muestre el nombre y el escudo de cada equipo, y además se genere el enlace para mostrar su información detallada.

En este caso se utiliza un ciclo @foreach de Blade.

```
@extends('layouts.base')
@section('title', 'Torneo')
@section('content')
  <div class="row">
    @foreach ($equipos as $key=>$equipo)
      <div class="col-3 text-center">
        <a href="/equipos/{{ $key }}">
          <img src="{{ $equipo['escudo'] }}">
          <h4>{{ $equipo['nombre'] }}</h4>
        </a>
      </div>
    @endforeach
  </div>
@endsection
```

Probar en el navegador.

El resultado debe quedar como el mostrado en la imagen.

Tarea

Crear un array de jugadores dentro de JugadoresController, con los campos foto, nombre, posición, número y equipo.

El campo posición puede ser Delantero, Volante, Defensa, Portero.

Modificar la vista index, de tal forma que muestre los datos de los jugadores en una tabla y a cada jugador le asigne un botón para ver detalle, otro para actualizar y otro para eliminar.

En la imagen se muestra un ejemplo de la distribución de los elementos.

Foto	Nombre	Posicion	Numero	Equipo	Accion
	James Rodriguez	Volante	3	Colombia	Ver Modificar Eliminar
	David Ospina	Portero	1	Colombia	Ver Modificar Eliminar

---

Actividad 10

A continuación, se van a realizar las acciones necesarias para mostrar la vista detalle de un equipo.

Inicialmente se modifica el método show de EquiposController, de tal forma que reciba el equipo y la posición del equipo en el array.

```php
public function show($id)
{
    return view('equipos.show')
                ->with('id', $id)
                ->with('equipo', $this->equipos[$id]);
}
```

En la vista show se agrega el código html en la sección content para que muestre el detalle del equipo seleccionado, y además tres botones, uno para eliminar, otro para modificar y otro para volver.

```blade
@extends('layouts.base')
@section('title', 'Detalle equipo')
@section('content')
    <div class="row">
        <div class="col-sm-4">
            <img src="{{ $equipo['escudo'] }}">
        </div>
        <div class="col-sm-8">
            <p class="h3">{{ $equipo['nombre'] }}</p>
            <p class="h5">{{ $equipo['dt'] }}</p>
            <p class="h5">{{ $equipo['municipio'] }}</p>
            <a href="#" class="btn btn-danger">Eliminar</a>
            <a href="/equipos/{{ $id }}/edit" class="btn btn-warning">Modificar</a>
            <a href="/equipos" class="btn btn-outline-dark">Ver equipos</a>
        </div>
    </div>
@endsection
```

Probar en el navegador.

El resultado debe ser similar al mostrado en la imagen.

Tarea

Modificar el método y la vista show de jugadores para mostrar la información detallada del jugador seleccionado en la tabla.

Formularios

Para crear un formulario se utilizan etiquetas de HTML, tanto para el formulario, como para sus elementos.

Es posible también utilizar directivas de Blade para construir el contenido HTML.

La acción del formulario apunta a una URL

HTML solo permite el uso de formularios de tipo GET o POST. Para realizar solicitudes de tipo PUT, PATCH o DELETE, se debe agregar un campo @method oculto para falsificar estos verbos HTTP. Esto se logra incluyendo la directiva de Blade @method, ejemplo. @method ('PUT')

Protección CSRF

Laravel facilita la protección frente a ataques de falsificación de solicitudes entre sitios (CSRF), generando automáticamente un "token" CSRF para cada sesión de usuario activa. Este token se usa para verificar que el usuario autenticado es el que realiza las solicitudes a la aplicación.

Este token se debe incluir en todos los formularios de la aplicación, usando la directiva @csrf de Blade.

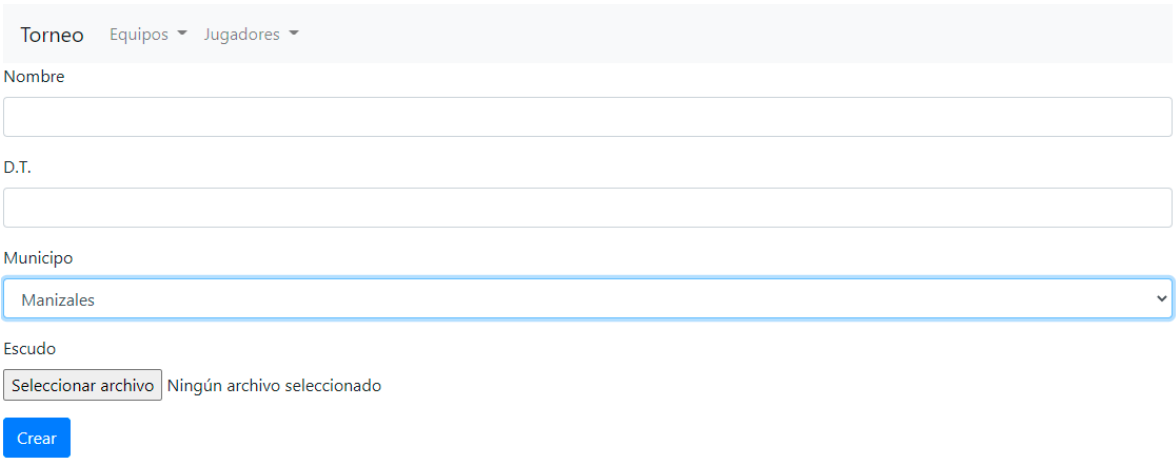
Actividad 11

Modificar la vista create de equipos insertando el código que aparece en la imagen, de tal forma que muestre el formulario para la creación de un nuevo equipo.

```
@extends('layouts.base')
@section('title', 'Crear equipo')
@section('content')
    <form action="/equipos" method="post" enctype="multipart/form-data">
        @csrf
        <div class="form-group">
            <label for="nombre">Nombre</label>
            <input type="text" name="nombre" id="nombre" class="form-control">
        </div>
        <div class="form-group">
            <label for="dt">D.T.</label>
            <input type="text" name="dt" id="dt" class="form-control">
        </div>
        <div class="form-group">
        <label for="municipio">Municipo</label>
        <select class="form-control" id="municipio" name="municipio">
            <option value="1">Manizales</option>
            <option value="2">Pereira</option>
            <option value="3">Armenia</option>
        </select>
    </div>
        <div class="form-group">
            <label for="escudo">Escudo</label>
            <input type="file" name="escudo" id="escudo" class="form-control-file">
        </div>
        <button type="submit" class="btn btn-primary">Crear</button>
    </form>
@endsection
```

Probar en el navegador.

El resultado debe ser similar al mostrado en la imagen.



Tarea

Crear un formulario para insertar un nuevo jugador.

Este debe tener los campos nombre (text), posición (select), numero (number), equipo (select) y foto (file).

El campo posición debe ser de tipo select, ya que puede ser Delantero, Volante, Defensa, Portero.

En el select de equipos se deben cargar tres equipos.

Base de datos

Laravel permite una interacción simple con bases de datos, utilizando el fluent query builder y Eloquent ORM. Actualmente, soporta MySQL, PostgreSQL, SQLite y SQLServer.

Actividad

En cualquier gestor de bases de datos MySQL, crear una base de datos llamada torneo.

---

para trabajar con bases de datos se debe configurar la conexión.

La configuración se encuentra en config/database.php, donde aparece la línea.

'default' => env('DB_CONNECTION', 'mysql'),

Este valor indica el tipo de base de datos a utilizar por defecto, obteniendo el valor de la variable DB_CONNECTION del fichero .env, en caso que no esté definida devolverá por defecto mysql.

Los valores se deben modificar en el archivo .env de la raíz del proyecto.

Actividad

Abrir el archivo .env de la raíz del proyecto, buscar los parámetros de configuración de mysql, y asignar los valores que se muestran en la imagen.

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=torneo
DB_USERNAME=root
DB_PASSWORD=
```

Migraciones

Las migraciones son un control de versiones para la base de datos, permiten modificar y compartir el esquema de la base de datos de la aplicación.

Las migraciones se crean en archivos PHP con la descripción de la tabla a crear y posteriormente, si se quiere modificar dicha tabla se añadiría una nueva migración con los campos a modificar. Artisan incluye comandos para crear migraciones, para ejecutar las migraciones o para hacer rollback de las mismas.

Crear migraciones

Para crear una nueva migración se utiliza el comando de Artisan make:migration, seguido del nombre del nombre del archivo.

php artisan make:migration create_users_table

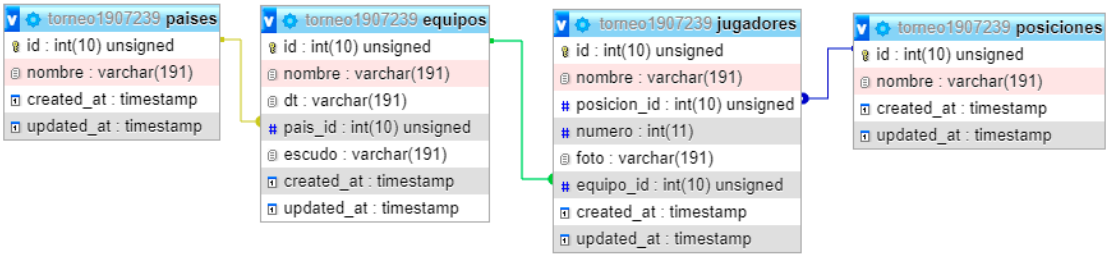
php artisan make:migration add_votes_to_user_table

La migración se almacenará en la carpeta database/migrations con un timestamp que permite conocer el orden en el que se deben ejecutar.

No hay restricción con el nombre del archivo, pero se recomienda seguir el patrón, create_<table-name>_table o <action>_<field>_to_<table-name>_table. Si se utiliza este patrón, Laravel crea la plantilla básica de la migración.

Actividad 11 Pendiente

Crear las migraciones para las tablas municipios, equipos, posiciones y jugadores. Debe ser en ese orden, ya que, de acuerdo con el modelo relacional mostrado en la tabla, la tabla equipos tiene una llave foránea hacia municipios, y jugadores se debe crear de ultima, ya que tiene llaves foráneas hacia equipos y posiciones.



Ejecutar los comandos mostrados en las imágenes para crear las migraciones. (Dar Ctrl + C en caso de que el servidor este en ejecución).

```
php artisan make:migration create_municipios_table
```

```
php artisan make:migration create_equipos_table
```

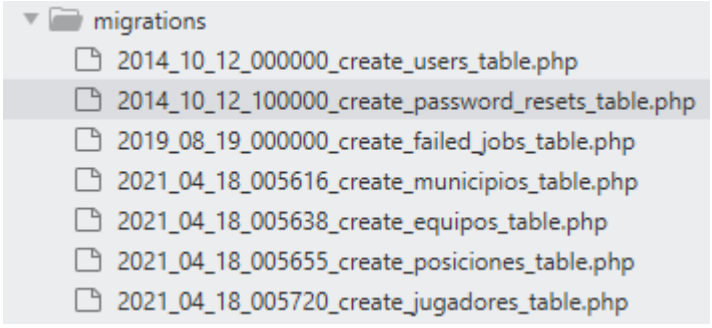
```
php artisan make:migration create_posiciones_table
```

```
php artisan make:migration create_jugadores_table
```

Una vez creada cada migración, se debe mostrar un mensaje conformándolo.

```
Created Migration: 2020_04_29_173640_create_equipos_table
```

Las migraciones se pueden encontrar en la carpeta database/migrations.



Adicional a las migraciones creadas, se pueden encontrar tres migraciones adicionales.

Estructura de la migración

Una migración contiene los métodos up y down.

El método up se usa para crear nuevas tablas, columnas, o índices en la base de datos, mientras que el método down debe reversar las operaciones realizadas por el método up.

En ambos métodos se puede usar Laravel schema builder para crear y modificar tablas.

```php
class CreateEquiposTable extends Migration
{
    public function up()
    {
        Schema::create('equipos', function (Blueprint $table) {
            $table->increments('id');
            $table->timestamps();
        });
    }
    public function down()
    {
        Schema::dropIfExists('equipos');
    }
}
```

Tablas

Para crear una tabla se utiliza el método create de Schema. Este método recibe el nombre de la tabla y una función donde se definen los elementos.

Para renombrar una tabla se utiliza el método rename.

Schema::rename($from, $to);

Para borrar una tabla existente se utilizan los métodos drop o dropIfExists

Schema::drop('users');

Schema::dropIfExists('users');

Tipos de columnas

La figura muestra los tipos de columnas que se pueden crear con Laravel.

bigIncrements	lineString	string
bigInteger	longText	text
binary	macAddress	timeTz
boolean	mediumIncrements	time
char	mediumInteger	timestampTz
dateTimeTz	mediumText	timestamp
dateTime	morphs	timestampsTz
date	multiLineString	timestamps
decimal	multiPoint	tinyIncrements
double	multiPolygon	tinyInteger
enum	nullableMorphs	tinyText
float	nullableTimestamps	unsignedBigInteger
foreignId	nullableUuidMorphs	unsignedDecimal
geometryCollection	point	unsignedInteger
geometry	polygon	unsignedMediumInteger
id	rememberToken	unsignedSmallInteger
increments	set	unsignedTinyInteger
integer	smallIncrements	uuidMorphs
ipAddress	smallInteger	uuid
json	softDeletesTz	year
jsonb	softDeletes	

Modificadores

En la imagen se muestran los modificadores que se pueden usar con Laravel.

Modifier	Description
`->after('column')`	Place the column "after" another column (MySQL).
`->autoIncrement()`	Set INTEGER columns as auto-incrementing (primary key).
`->charset('utf8mb4')`	Specify a character set for the column (MySQL).
`->collation('utf8mb4_unicode_ci')`	Specify a collation for the column (MySQL/PostgreSQL/SQL Server).
`->comment('my comment')`	Add a comment to a column (MySQL/PostgreSQL).
`->default($value)`	Specify a "default" value for the column.
`->first()`	Place the column "first" in the table (MySQL).
`->from($integer)`	Set the starting value of an auto-incrementing field (MySQL / PostgreSQL).
`->nullable($value = true)`	Allow NULL values to be inserted into the column.
`->storedAs($expression)`	Create a stored generated column (MySQL).
`->unsigned()`	Set INTEGER columns as UNSIGNED (MySQL).
`->useCurrent()`	Set TIMESTAMP columns to use CURRENT_TIMESTAMP as default value.
`->useCurrentOnUpdate()`	Set TIMESTAMP columns to use CURRENT_TIMESTAMP when a record is updated.
`->virtualAs($expression)`	Create a virtual generated column (MySQL).
`->generatedAs($expression)`	Create an identity column with specified sequence options (PostgreSQL).
`->always()`	Defines the precedence of sequence values over input for an identity column (PostgreSQL).

Crear índices

La figura muestra la forma en la cual Laravel permite crear índices.

Command	Description
`$table->primary('id');`	Adds a primary key.
`$table->primary(['id', 'parent_id']);`	Adds composite keys.
`$table->unique('email');`	Adds a unique index.
`$table->index('state');`	Adds an index.
`$table->spatialIndex('location');`	Adds a spatial index (except SQLite).

Longitud de índices

Laravel usa por defecto utf8mb4 que incluye soporte para almacenar emojis en la base de datos. si se está ejecutando una versión de MySQL inferior a la 5.7.7 o MariaDB inferior a 10.2.2 se debe modificar manualmente la longitud de strings.

Esto se puede hacer llamando el método Schema::defaultStringLength(191); dentro del método boot en app/Providers/AppServiceProvider

---

Actividad 12

Con el fin de evitar errores al momento de crear las tablas en la base de datos, se van a modificar manualmente la longitud de los strings.

app/Providers/AppServiceProvider.php

Agregar el código para incluir Schema.

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

En el método boot incluir la instrucción mostrada en la imagen.

```
public function boot()
{
    Schema::defaultStringLength(191);
}
```

Llaves foráneas

Las imágenes mostradas a continuación muestran la forma en la cual se crear llaves foráneas.

```
Schema::table('posts', function (Blueprint $table) {
    $table->unsignedBigInteger('user_id');

    $table->foreign('user_id')->references('id')->on('users');
});
```

Esta sintaxis es muy detallada, Laravel proporciona métodos más breves que utilizan convenciones para proporcionar una mejor experiencia de desarrollador. El ejemplo anterior se puede reescribir así:

```
Schema::table('posts', function (Blueprint $table) {
    $table->foreignId('user_id')->constrained();
});
```

El método foreignId() es un alias para unsignedBigInteger, mientras que el método constrained() usará convenciones para determinar la tabla y el nombre de la columna a la que se hace referencia. Si el nombre de su tabla no coincide con las

convenciones de Laravel, se puede especificar el nombre de la tabla pasándolo como un argumento al método constrained().

```
Schema::table('posts', function (Blueprint $table) {
    $table->foreignId('user_id')->constrained('users');
});
```

También se puede especificar la acción deseada para las propiedades "al eliminar" y "al actualizar".

```
$table->foreignId('user_id')
      ->constrained()
      ->onUpdate('cascade')
      ->onDelete('cascade');
```

Actividad 13

A continuación, se crearán las migraciones para las tablas municipios, equipos, posiciones y jugadores.

Abrir el archivo de migración de municipios ubicado en la carpeta database/migrations, y modificar el código como se muestra en la imagen.

```php
<?php
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;
class CreateMunicipiosTable extends Migration
{
    public function up()
    {
        Schema::create('municipios', function (Blueprint $table) {
            $table->id();
            $table->string('nombre', 100);
            $table->timestamps();
        });
    }
    public function down()
    {
        Schema::dropIfExists('municipios');
    }
}
```

Abrir el archivo de migración de equipos ubicado en la carpeta database/migrations, y modificar el código como se muestra en la imagen.

```php
<?php
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class CreateEquiposTable extends Migration
{
    public function up()
    {
        Schema::create('equipos', function (Blueprint $table) {
            $table->id();
            $table->string('nombre', 100);
            $table->string('dt', 100);
            $table->foreignId('municipio_id')->constrained();
            $table->string('escudo', 100);
            $table->timestamps();
        });
    }
    public function down()
    {
        Schema::dropIfExists('equipos');
    }
}
```

Abrir el archivo de migración de posiciones ubicado en la carpeta database/migrations, y modificar el código como se muestra en la imagen.

```php
<?php
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;
class CreatePosicionesTable extends Migration
{
    public function up()
    {
        Schema::create('posiciones', function (Blueprint $table) {
            $table->id();
            $table->string('nombre', 100);
            $table->timestamps();
        });
    }
    public function down()
    {
        Schema::dropIfExists('posiciones');
    }
}
```

Abrir el archivo de migración de jugadores ubicado en la carpeta database/migrations, y modificar el código como se muestra en la imagen.

```php
<?php
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;
class CreateJugadoresTable extends Migration
{
    public function up()
    {
        Schema::create('jugadores', function (Blueprint $table) {
            $table->id();
            $table->string('nombre', 100);
            $table->foreignId('posicion_id')->constrained('posiciones');
            $table->integer('numero');
            $table->string('foto', 100);
            $table->foreignId('equipo_id')->constrained();
            $table->timestamps();
        });
    }
    public function down()
    {
        Schema::dropIfExists('jugadores');
    }
}
```

Ejecutar migraciones

Para ejecutar todas las migraciones se utiliza el comando.

php artisan migrate

Actividad 14

Ejecutar las migraciones para crear las tablas de la base de datos.

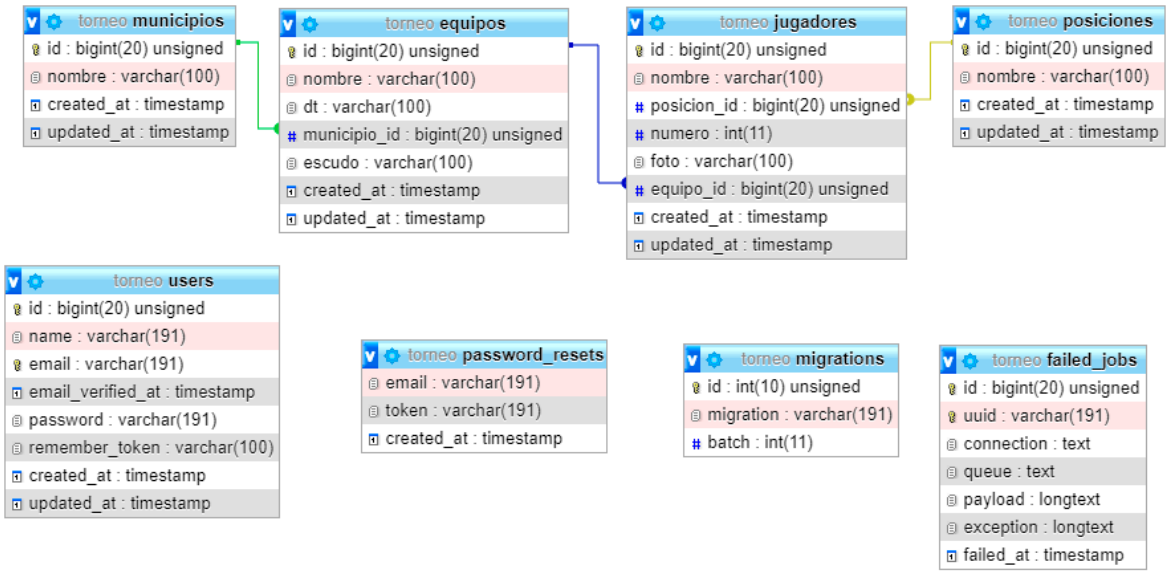
En cmd ejecutar el comando mostrado en la imagen.

```
php artisan migrate
```

Si todo salió bien, se muestra un mensaje similar al mostrado en la imagen.

```
Migration table created successfully.
Migrating: 2014_10_12_000000_create_users_table
Migrated:  2014_10_12_000000_create_users_table (704.46ms)
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated:  2014_10_12_100000_create_password_resets_table (520.09ms)
Migrating: 2019_08_19_000000_create_failed_jobs_table
Migrated:  2019_08_19_000000_create_failed_jobs_table (517.93ms)
Migrating: 2021_04_18_005616_create_municipios_table
Migrated:  2021_04_18_005616_create_municipios_table (219.66ms)
Migrating: 2021_04_18_005638_create_equipos_table
Migrated:  2021_04_18_005638_create_equipos_table (1,396.19ms)
Migrating: 2021_04_18_005655_create_posiciones_table
Migrated:  2021_04_18_005655_create_posiciones_table (405.93ms)
Migrating: 2021_04_18_005720_create_jugadores_table
Migrated:  2021_04_18_005720_create_jugadores_table (4,406.74ms)
```

Al verificar la base de datos en el gestor de bases de datos, se puede verificar que se crearon las tablas municipios, equipos, posiciones y jugadores con sus respectivas relaciones, además se crearon cuatro tablas adicionales.



Revertir migraciones

Para revertir la última migración, se usa el comando rollback, que regresa al último "batch" de migraciones.

php artisan migrate:rollback

El comando migrate:reset deshará todas las migraciones.

php artisan migrate:reset

El comando migrate:refresh revertirá todas tus migraciones y luego ejecutará el comando migrate

php artisan migrate:refresh

El comando migrate:fresh eliminará todas las tablas de la base de datos y luego ejecutará el comando migrate.

php artisan migrate:fresh