PROYECTO LARAVEL

Primero activamos nuestro servidor XAMPP. Luego creamos una carpeta donde guardaremos nuestro proyecto, abrimos el CMD, y navegamos hasta la carpeta htdocs ubicada en XAMPP, en esta colocaremos nuestro proyecto.

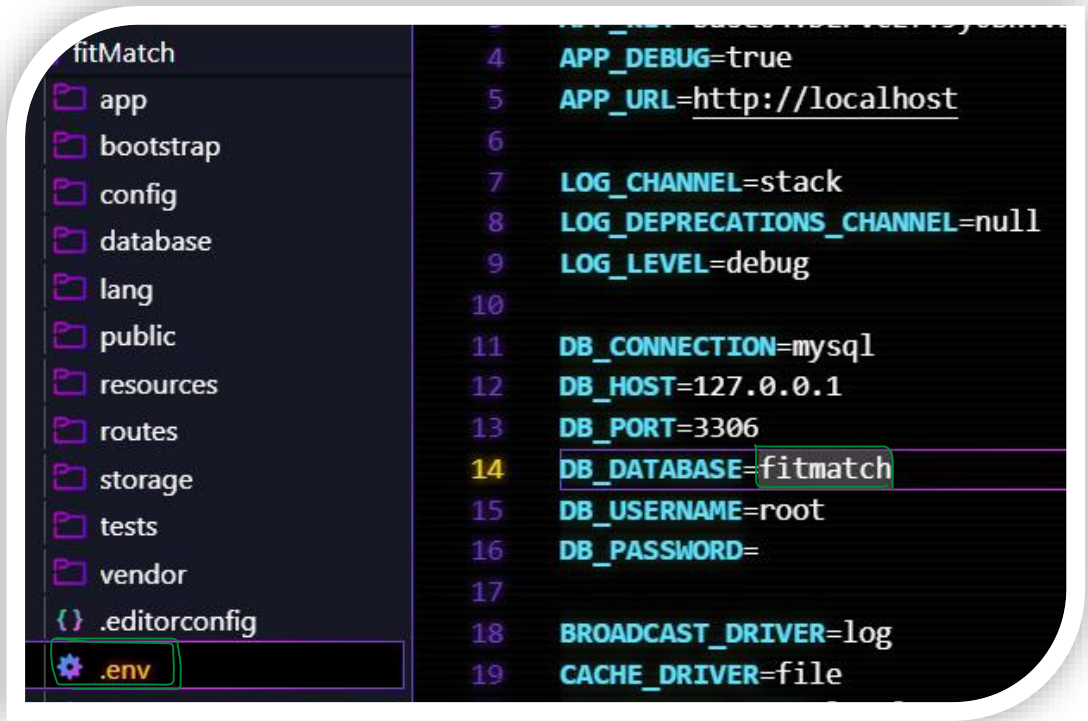
Para crear un nuevo proyecto de Laravel, ejecutaremos el siguiente comando:

- composer create-project laravel/laravel nombre_proyecto "versión laravel"

```
C:\xampp\htdocs\fitmatch>composer create-project laravel/laravel fitMatch "9."
Creating a "laravel/laravel" project at "./fitMatch"
Installing laravel/laravel (v9.0.0)
  - Installing laravel/laravel (v9.0.0): Extracting archive
Created project in C:\xampp\htdocs\fitmatch\fitMatch
> @php -r "file_exists('.env') || copy('.env.example', '.env');"
```

Laravel generará automáticamente una serie de archivos y carpetas dentro del directorio, y será la estructura base de nuestro proyecto, encontramos un archivo .env , que contiene variables de configuración para la base de datos y el servidor local.

Tendremos que cambiar DB_DATABASE="nombre_base_de_datos", ya que por defecto viene laravel y además de crearlo en nuestro gestor de bases datos.



Modificaremos el archivo composer.json , añadimos "appzcoder/crud-generator":"^3.2" en la sección de "require-dev".

Ejecutamos para ver si creo correctamente: nombre_proyecto>php artisan serve (nos saldrá la página de laravel)

Creación de migraciones:

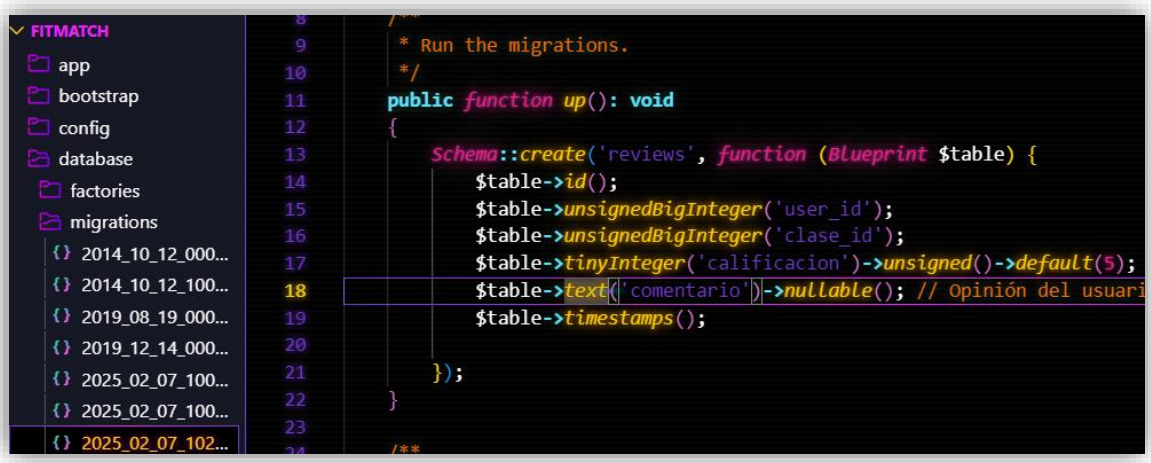
Primero se crear la tabla que no contiene el campo relacional, después las demás, es importante seguir el orden.

Ejecutaremos el comando de >php artisan make:migration create_nombreTabla_table

```
C:\xampp\htdocs\fitmatch\fitmatch>php artisan make:migration create_clases_table

  INFO  Migration [C:\xampp\htdocs\fitmatch\fitmatch\database\migrations/2025_02_07_100158_create_clases_table.php] created successfully.
```

Una vez creada las tablas pasamos a editar los campos según queramos...



Por último, ya editado todo, abrimos la consola y ejecutamos el comando >php artisan migrate

Si por alguna razón quieres cambiar algo de nuevo >php artisan migrate: refresh

```
C:\xampp\htdocs\fitmatch\fitmatch>php artisan migrate

  INFO  Preparing database.

Creating migration table ..... 44ms DONE

  INFO  Running migrations.

2014_10_12_000000_create_users_table ..... 35ms DONE
2014_10_12_100000_create_password_reset_tokens_table ..... 4ms DONE
2019_08_19_000000_create_failed_jobs_table ..... 35ms DONE
2019_12_14_000001_create_personal_access_tokens_table ..... 17ms DONE
2025_02_07_100158_create_clases_table ..... 4ms DONE
2025_02_07_100247_create_suscripciones_table ..... 4ms DONE
2025_02_07_102027_create_reviews_table ..... 4ms DONE
2025_02_07_105549_create_eventos_table ..... 5ms DONE
```

Autenticación:

Utilizaremos ui con Bootstrap, también hay otras... Los comandos que tendremos que ejecutar:

Proyecto>composer require laravel/ui

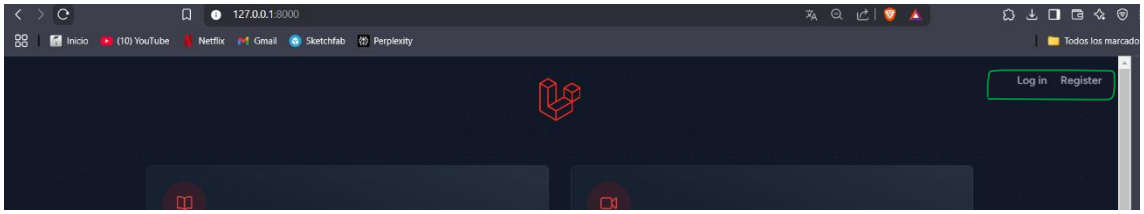
```
:\xampp\htdocs\fitmatch\fitmatch>composer require laravel/ui
/composer.json has been updated
unning composer update laravel/ui
oading composer repositories with package information
pdating dependencies
ock file operations: 3 installs, 0 updates, 0 removals
- Locking appzcoder/crud-generator (v3.3.0)
```

Proyecto>php artisan ui boostrap --auth

Proyecto>npm install

Proyecto>npm run dev

IMPORTANTE : el último comando hay que ejecutarlo en otra terminal y quede de forma activa



Al lanzarlo se nos añadirá dos nuevas secciones, y nos registramos, donde los datos registrados se guardaran en la tabla user.

Creación y Uso de los Cruds.

Ejecutamos en el cmd, teneindo activo el npm .

Proyecto>composer require lbex/crud-generator –dev

Proyecto>php artisan vendor:publish --tag=crud

Proyecto>php artisan make:crud nombre_tabla

```
PS C:\xampp\htdocs\fitmatch\fitmatch> php artisan make:crud suscripciones

  Which stack would you like to install?
  Blade with Bootstrap css ..... bootstrap
  Blade with Tailwind css ..... tailwind
  Livewire with Tailwind css ..... livewire
  API only ..... api
› bootstrap
```

Cuando ejecutamos el make, escribimos Bootstrap y ya nos crea el modelo, controlador y las vistas de nuestro proyecto

Acceso a los Cruds

Cuando hemos generado los crud, tenemos que establecer una ruta a cada uno de ellos, para ello, nos posicionamos en la carpeta Routes/web.php , y escribimos las siguientes rutas de tipo Resource

```php
Auth::routes();
Route::resource('clases', App\Http\Controllers\ClaseController::class)->middleware('auth');
Route::resource('reviews', App\Http\Controllers\ReviewController::class)->middleware('auth');
Route::resource('eventos', App\Http\Controllers\EventoController::class)->middleware('auth');
Route::resource('suscripciones', App\Http\Controllers\SuscripcioneController::class)->middleware('auth');
Route::get('/home', [App\Http\Controllers\HomeController::class, 'index'])->name('home');
```

Creación de los enlaces tablas

Al ejecutar el comando de crud-generator se nos han creado los archivos básicos para tener un crud funcional, incluidas las vistas. Ahora podremos modificar el código, vamos a añadir un menú en nuestro proyecto para acceder a nuestras tablas

Para ello vamos resources\views\layouts\app.blade.php y creamos los enlaces.

En mi caso lo pegare en mi menú(nav), y ya he hecho los roles que mas adelante explicare.

```blade
<!-- Navbar -->
<nav class="navbar">
    <h1 class="logo">FitMatch</h1>
    <ul class="navbar-links">
        <li><a href="#hero" class="nav-link">Inicio</a></li>
        <li><a href="#reviews" class="nav-link">Opiniones</a></li>
        @auth
            <li><a href="{{ url('/home') }}" class="nav-link">Mi cuenta</a></li>
            @if (Auth::user()->role === 'admin')
                <li><a href="{{ route('classes.index') }}" class="nav-link">Gestionar Clases</a></
                <li><a href="{{ route('coaches.index') }}" class="nav-link">Gestión de Entrenadore
                {{-- <li><a href="{{ route('reservations.index') }}" class="nav-link">Reservas</a>
            @endif
            <li>
                <form action="{{ route('logout') }}" method="POST">
                    @csrf
                    <button type="submit" id="btnLogOut" class="nav-link logout">Cerrar sesión</bu
                </form>
            </li>
        @else
            <li><a href="{{ route('login') }}" class="nav-link">Iniciar sesión</a></li>
            @if (Route::has('register'))
                <li><a href="{{ route('register') }}" class="nav-link">Registrarse</a></li>
            @endif
        @endauth
    </ul>
</nav>
```

Acceso a los datos de las tablas.

Para poder acceder a los datos, debo poder llamar desde una clase a sus datos, para ello modificamos los métodos de nuestro controlador:

```php
/**
 * Show the form for creating a new resource.
 */
0 references | 0 overrides
public function create()
{
    $coaches = Coach::all();

    return view('classes.create', compact('coaches'));
}
```

```php
/**
 * Show the form for editing the specified resource.
 */
0 references | 0 overrides
public function edit(string $id)
{
    $class = ClassModel::findOrFail($id);
    $coaches = Coach::all();

    return view('classes.edit', compact('class', 'coaches'));
}
```

Realizar el formulario.

@extends-> indica que plantilla hereda de un layout principal.

@section-> define el bloque de contenido que se inyectará en la sección del content del layout principal

@if (session)... -> a partir de ahí manejamos mensajes de éxito y errores

```
@extends('layouts.app')

@section('content')
    <div class="container">
        <h1>Crear Nueva Clase</h1>

        {{-- Mostrar mensajes de éxito o error --}}
        @if (session('success'))
            <div class="alert alert-success">{{ session('success') }}</div>
        @endif

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

        <form action="{{ route('classes.store') }}" method="POST">
            @csrf

            {{-- Nombre de la clase --}}
            <div class="mb-3">
                <label for="name" class="form-label">Nombre de la clase</label>
                <input type="text" name="name" id="name" class="form-control
@error('name') is-invalid @enderror"
                    value="{{ old('name') }}" placeholder="Ej: Zumba, Yoga,
etc.">
                @error('name')
                    <div class="invalid-feedback">{{ $message }}</div>
                @enderror
            </div>

            {{-- Descripción --}}
            <div class="mb-3">
                <label for="description" class="form-label">Descripción</label>
                <textarea name="description" id="description" rows="3"
                    class="form-control @error('description') is-invalid
@enderror" placeholder="Breve descripción de la clase">{{ old('description')
}}</textarea>
                @error('description')
                    <div class="invalid-feedback">{{ $message }}</div>
                @enderror
            </div>
```

```
            {{-- Fecha y hora --}}
            <div class="mb-3">
                <label for="date_time" class="form-label">Fecha y Hora</label>
                <input type="datetime-local" name="date_time" id="date_time"
                    class="form-control @error('date_time') is-invalid @enderror"
value="{{ old('date_time') }}">
                @error('date_time')
                    <div class="invalid-feedback">{{ $message }}</div>
                @enderror
            </div>
            <div class="mb-3">
                <label for="duration" class="form-label">Duración
(minutos)</label>
                <input type="number" name="duration" id="duration"
                    class="form-control @error('duration') is-invalid @enderror"
value="{{ old('duration') }}">
                @error('duration')
                    <div class="invalid-feedback">{{ $message }}</div>
                @enderror
            </div>
            {{-- Capacidad --}}
            <div class="mb-3">
                <label for="capacity" class="form-label">Capacidad</label>
                <input type="number" name="capacity" id="capacity"
                    class="form-control @error('capacity') is-invalid @enderror"
value="{{ old('capacity') }}"
                    placeholder="Ej: 20">
                @error('capacity')
                    <div class="invalid-feedback">{{ $message }}</div>
                @enderror
            </div>

            {{-- Instructor (opcional) --}}
            <div class="mb-3">
                <label for="coach_id" class="form-label">Entrenador</label>
                <select name="coach_id" id="coach_id" class="form-control">
                    <option value="">Selecciona un entrenador</option>
                    @foreach ($coaches as $coach)
                        <option value="{{ $coach->id }}">{{ $coach->name
}}</option>
                    @endforeach
                </select>
            </div>


            {{-- Botón de enviar --}}
            <button type="submit" class="btn btn-primary">Guardar Clase</button>

            {{-- Botón de regresar al listado --}}
            <a href="{{ route('classes.index') }}" class="btn btn-
secondary">Volver</a>
        </form>
    </div>
```

```
@endsection
```

Modificación del nombre del encabezado, aquí simplemente debemos saber que puedo acceder $clase->tabla->nombre

(YA QUE HEMOS IMPLENTADO UN MÉTODO)

Pasos para crear un middleware de administrador

Creamos el middleware:

```
PS C:\xampp\htdocs\proyecto_fitmatch\fitmatch> php artisan make:middleware AdminMiddleware
```

Esto nos generará app/Http/Middleware/AdminMiddleware.php

Pasaremos a editarlo:

```php
namespace App\Http\Middleware;

use Closure;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;

class AdminMiddleware
{
    public function handle(Request $request, Closure $next)
    {
        if (!Auth::check() || !Auth::user()->is_admin) {
            abort(403, 'No tienes permiso para acceder aquí');
        }

        return $next($request);
    }
}
```

Registrar el middleware en app/Http/Kernel.php, dentro de $middlewareAliases, agregamos la clave 'admin':

protected $middlewareAliases = [

//resto de rutas

'admin' => \App\Http\Middleware\AdminMiddleware::class, // <-- ¡AQUÍ!

];

Proteger las rutas en routes/web.php

```php
Route::middleware(['auth', 'admin'])->group(function () {
    Route::resource('coaches', CoachController::class);
    Route::resource('classes', ClaseController::class);
});
```

Donde queramos gestionar nuestros botones, es decir que solo el administrador pueda usarlo vamos modificar en nuestra vista que usemos esos botones y pondremos lo siguiente:

```
@if (Route::has('login'))
    <div class="sm:fixed sm:top-0 sm:right-0 p-6 text-right">
        @auth
            <a href="{{ url('/home') }}"
                class="font-semibold text-gray-600 hover:text-gray-900 dark:text
            @if (Auth::user()->role === 'admin')
                <a href="{{ route('classes.index') }}" class="btn btn-primary">Gestiona

                <a href="{{ route('coaches.index') }}" class="btn btn-primary">Gestión
            @endif
```

Para asignar un usuario como administrador, usamos el comando php artisan tinker. , y metemos el correo de nuestro usuario que sea administrador

```
PS C:\xampp\htdocs\proyecto_fitmatch\fitmatch> php artisan tinker
Psy Shell v0.12.7 (PHP 8.2.12 — cli) by Justin Hileman
> $user = App\Models\User::where('email', 'adminfitmatch@fitmatch.com')->first();
```

Ahora tenemos que asignarle el rol de administrador para ello ejecutamos el siguiente comando:

```
$user->role = 'admin'; $user->save();
```

Una vez hecho hacemos un exit y listo.

Para obtener las traducciones en español, para ello vamos a resources, creamos nuestra carpeta Lang y desde el repositorio de github, lo vamos clonar con el siguiente comando:

```
git clone https://github.com/Laravel-Lang/lang.git
```

Una vez clonado, abrimos el archivo config/app.php , y buscamos la línea: "locale" => "en", lo cambiamos en nuestro caso por "es", porque lo queremos en español.

ANEXO

Opcional, modificación de css, js... , para usar nuestros estilos, librerías podemos añadirlas en la carpeta de public, para poder acceder a ellas desde el php :

En cambio, si necesitamos desde nuestro propio css algo de la carpeta public no haría falta, seria con ruta relativa.

Anotación→ Posibles mejoras futuras (por ejemplo, enviar emails de confirmación de reserva, panel de administración más avanzado, etc.).