

PROYECTO LARAVEL

1. Creación del Proyecto

1.1 Preparación del Entorno

Para comenzar a trabajar con Laravel, primero debemos activar nuestro servidor XAMPP. Posteriormente, creamos una carpeta donde almacenaremos nuestro proyecto. Usamos el CMD para navegar hasta la carpeta htdocs ubicada en el directorio de instalación de XAMPP y ahí es donde colocaremos nuestro proyecto.

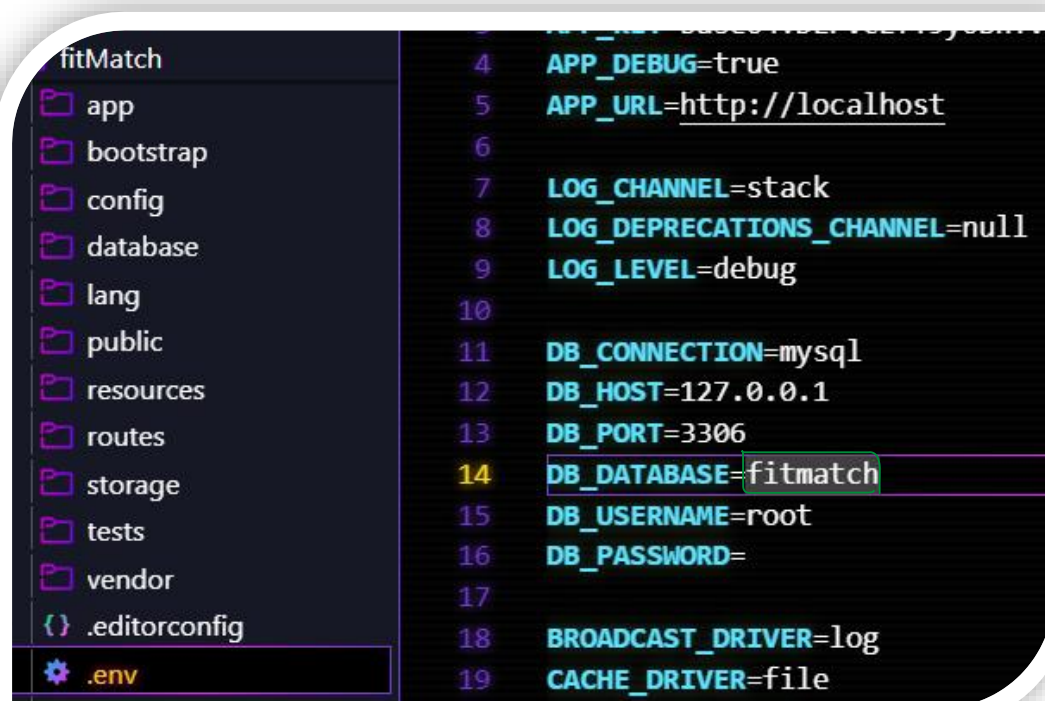
1.2 Instalación de Laravel

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');"
```

Este comando descargará e instalará Laravel en el directorio especificado, creando la estructura base del proyecto. Uno de los archivos más importantes es el archivo .env, donde configuramos las variables de entorno, especialmente las relacionadas con la base de datos. Aquí debemos cambiar `DB_DATABASE="nombre_base_de_datos"` para reflejar el nombre de nuestra base de datos, que previamente debe ser creada en nuestro gestor de base de datos.



Para instalar una dependencia adicional, debemos modificar el archivo `composer.json` y añadir lo siguiente en la sección `"require-dev"`:

```
→ "require-dev": {  
    "appzccoder/crud-generator": "^3.2",    n/a  
    "fakerphp/faker": "^1.9.1",    v1.24.1  
    "laravel/pint": "^1.0",    v1.20.0  
    "laravel/sail": "^1.18",    v1.41.0  
    "mockery/mockery": "^1.4.4",    v1.6.12  
    "nunomaduro/collision": "^7.0",    v7.11.0  
    "phpunit/phpunit": "^10.0",    v10.5.44  
    "spatie/laravel-ignition": "^2.0"    v2.9.0  
}
```

2 Estructura del Proyecto

2.1 Controladores

Los controladores son los encargados de manejar las peticiones HTTP y devolver las respuestas adecuadas. Algunos controladores importantes en este proyecto son:

- `ClassController`: Gestiona las clases disponibles para reserva.
- `ReservationController`: Maneja las reservas de las clases.
- `UserController`: Administra los usuarios, con acceso restringido solo a administradores.

2.2 Modelos

Los modelos representan las tablas de la base de datos y gestionan la interacción con los datos. Modelos clave incluyen:

- `ClassModel`: Modelo para las clases de fitness, con relaciones a las reservas y al instructor.
- `Reservation`: Modelo que maneja las reservas realizadas por los usuarios.
- `User`: Modelo para los usuarios, que incluye roles y permisos.

2.3 Rutas

Las rutas se definen en el archivo `routes/web.php`. Algunas rutas importantes incluyen:

- `GET /classes`: Muestra todas las clases disponibles.
- `GET /classes/{id}`: Muestra los detalles de una clase específica.
- `POST /reservations`: Crea una nueva reserva.
- `DELETE /reservations/{id}`: Cancela una reserva.

2.4 Vistas

Las vistas se encuentran en `resources/views`, donde gestionamos las interfaces de usuario. Ejemplos de vistas:

- `classes.index`: Muestra la lista de todas las clases.
- `reservations.index`: Muestra las reservas realizadas por el usuario o administrador.
- `coaches.show`: Muestra los detalles de un entrenador.

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

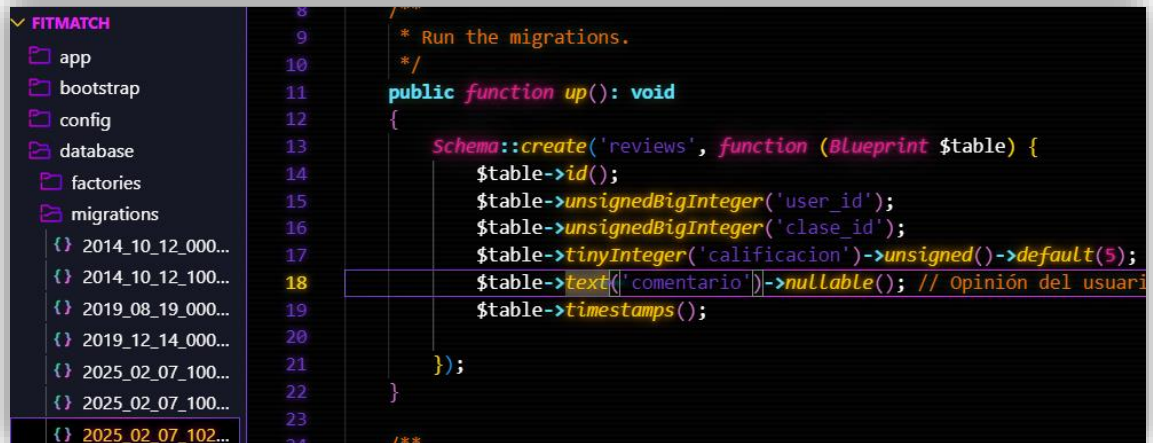
Creación de migraciones:

Primero se crea 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
```

Uso Seeder

Los seeders permiten poblar la base de datos con datos iniciales de manera automática. Esto es útil para pruebas, desarrollo y despliegue, ya que facilita la creación de registros sin necesidad de ingresarlos manualmente.

Implementación en el Proyecto

En nuestro caso, hemos utilizado un seeder en el archivo DatabaseSeeder.php, ubicado en la carpeta database/seeder/.

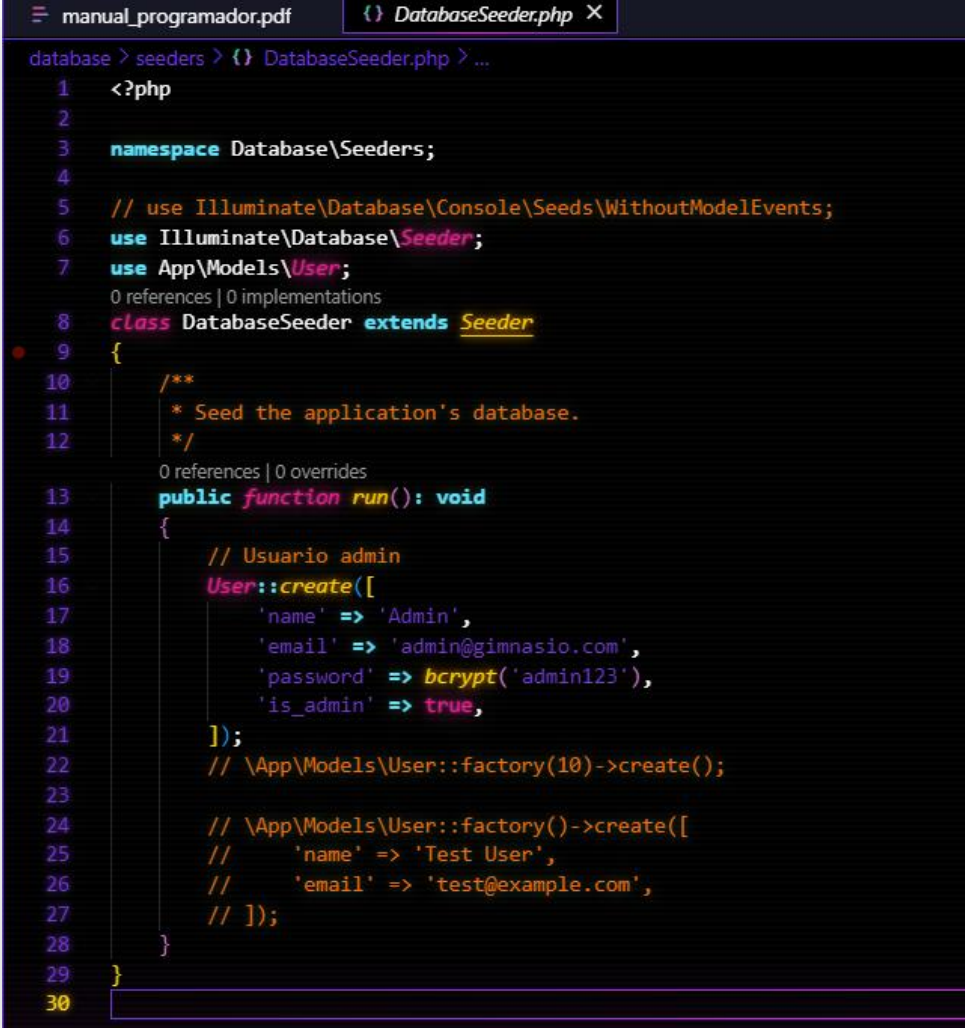
Código del Seeder

El código del seeder en nuestro proyecto está configurado para crear un usuario administrador con las siguientes credenciales

Además, ejecutamos los siguientes comandos:

```
>php artisan db:seed
```

```
>php artisan migrate:refresh --seed
```



```
manual_programador.pdf DatabaseSeeder.php X
database > seeders > DatabaseSeeder.php > ...
1  <?php
2
3  namespace Database\Seeders;
4
5  // use Illuminate\Database\Console\Seeds\WithoutModelEvents;
6  use Illuminate\Database\Seeder;
7  use App\Models\User;
8  class DatabaseSeeder extends Seeder
9  {
10     /**
11      * Seed the application's database.
12      */
13     0 references | 0 overrides
14     public function run(): void
15     {
16         // Usuario admin
17         User::create([
18             'name' => 'Admin',
19             'email' => 'admin@gimnasio.com',
20             'password' => bcrypt('admin123'),
21             'is_admin' => true,
22         ]);
23         // \App\Models\User::factory(10)->create();
24
25         // \App\Models\User::factory()->create([
26             //     'name' => 'Test User',
27             //     'email' => 'test@example.com',
28             // ]);
29     }
30 }
```

Autenticación:

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

Proyecto>composer require laravel/ui

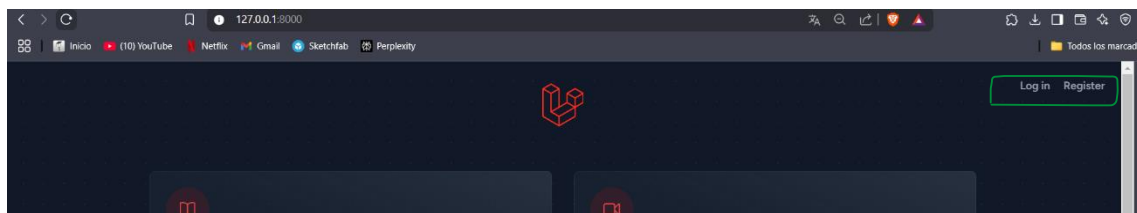
Proyecto>php artisan ui bootstrap --auth

Proyecto>npm install

Proyecto>npm run dev

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

```
C:\xampp\htdocs\fitmatch\fitmatch>composer require laravel/ui
./composer.json has been updated
Running composer update laravel/ui
Loading composer repositories with package information
Updating dependencies
Lock file operations: 3 installs, 0 updates, 0 removals
- Locking appzcode/crud-generator (v3.3.0)
```



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

```
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\SuscripcionController::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.

```
<!-- 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>
        <li><a href="{{ route('coaches.index') }}" class="nav-link">Gestión de Entrenadores</a></li>
        <!-- <li><a href="{{ route('reservations.index') }}" class="nav-link">Reservas</a></li>
      @endif
      <li>
        <form action="{{ route('logout') }}" method="POST">
          @csrf
          <button type="submit" id="btnLogOut" class="nav-link logout">Cerrar sesión</button>
        </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:

```
/**
 * Show the form for creating a new resource.
 */
0 references | 0 overrides
public function create()
{
    $coaches = Coach::all();
    return view('classes.create', compact('coaches'));
}
```

```
/**
 * 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 IMPLEMENTADO UN MÉTODO)

Pasos para crear un middleware de administrador

Creemos 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:

```
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`

```
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 a 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-gray-500">Inicio
    @if (Auth::user()->role === 'admin')
    <a href="{{ route('classes.index') }}" class="btn btn-primary">Gestión de Clases
    <a href="{{ route('coaches.index') }}" class="btn btn-primary">Gestión de Entrenadores
    @endif
  @endauth
</div>
```

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](https://github.com/Laravel-Lang/lang), creamos nuestra carpeta `Lang` y desde el repositorio de github, lo vamos a 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.

5. Recomendaciones para Desarrolladores

5.1 Buenas Prácticas

- Usa migration para gestionar la estructura de la base de datos.
- Utiliza seeders para poblar la base de datos con datos de prueba.
- Eloquent ORM es la herramienta recomendada para interactuar con la base de datos.
- No olvides validar los datos que provienen de los formularios utilizando el sistema de validación de Laravel.

5.2 Mantenimiento

- Mantén las dependencias actualizadas con el comando `composer update`.
- Revisa periódicamente las migraciones para asegurar que la estructura de la base de datos sea coherente con los cambios en el sistema.

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 :

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

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

Para modificar el css que nos genera el laravel, es decir poner el nuestro simplemente vamos app.blade.php , cambiamos la ruta o la hacemos directamente donde esta.

```
<link href="https://fonts.bunny.net/css?family=Nunito" rel="stylesheet">

<!-- Scripts -->
@vite(['resources/css/app.css', 'resources/js/app.js'])
</head>

<body>
<div id="app">
<nav class="navbar navbar-expand-md navbar-light bg-white shadow-sm">
<div class="container">
<a class="navbar-brand" href="{{ url('/') }}">
    {{ config('app.name', 'Laravel') }}
</a>
<button class="navbar-toggler" type="button" data-bs-toggle="collapse"
    data-bs-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">
    <!-- Left Side Of Navbar -->
```

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