Nuevo:
Nos situamos en la carpeta htdocs, donde si usamos xampp estamos atados por defecto a usar ese directorio para nuestras webs

```
cd C:/xampp/htdocs/
```

Creamos una nueva app, en este caso el tutorial la llama crudposts:
`laravel new crudposts` Respondemos breeze, blade, no, 0, mysql

Nos movemos al directorio de la app creada:

```
cd crudposts
```

Creamos la base de datos:
`php artisan migrate` Nos dice no existe base crudposts, la creo?, respondemos yes

Creamos una tabla llamada posts:

```
php artisan make:migration create_posts_table
```

Abrimos vscode:

```
code .
```

Nos dirigimos a database->migrations veremos un archivo 2023….create_posts_table.php lo editamos para agregar los campos(titulo de columnas) de la tabla, estos agregados se ponen luego de id y antes de timestamps:

```
            $table->id();
            $table->string('title');
            $table->text('body');
            $table->timestamps();
```

Salvamos.

Migramos a la base de datos el cambio a la tabla posts:

```
php artisan migrate
```

Creamos el controlador pero que nos agregue todos los métodos de CRUD, estarán vacíos, pero ahorra tiempo:

```
php artisan make:controller PostController -r
```

Llenamos los métodos, el primero será store, porque así lo dice el tutorial, podríamos empezar por cualquiera si supiesemos:

El método store:

```
public function store(Request $request)
    {

       $request->validate([
          'title' => 'required|max:255',
          'body' => 'required',
       ]);
       Post::create($request->all());
       return redirect()->route('posts.index')->with('success','Post created
successfully.');
}
```

El método index:

```
public function index()
    {
        $posts = Post::all();

        return view('posts.index', compact('posts'));
    }
```

El método update:

```
public function update(Request $request, $id)
    {
        $request->validate([
            'title' => 'required|max:255',
            'body' => 'required',
        ]);

        $post = Post::find($id);
        $post->update($request->all());

        return redirect()->route('posts.index')
            ->with('success', 'Post updated successfully.');
    }
```

El método destroy:

```
public function destroy($id)
    {

        $post = Post::find($id);
        $post->delete();
```

```
        return redirect()->route('posts.index')
            ->with('success', 'Post deleted successfully');
    }
```

El método create:

```
public function create()
    {
        return view('posts.create');
    }
```

El método show:

```
public function show($id)
    {
        $post = Post::find($id);

        return view('posts.show', compact('post'));
    }
```

El método edit:

```
public function edit($id)
    {
        $post = Post::find($id);

        return view('posts.edit', compact('post'));
    }
```

En el archivo el orden es index, store, update, destroy, create, show. y como devuelven vistas, se agrega el *use* del modelo en la parte superior, en el archivo app/Http/Controller/PostController:

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\Post;

class PostController extends Controller{
```

Configurar y crear el Modelo:
El modelo Post interactúa con la tabla posts de la base de datos.
Creamos:

```
php artisan make:model Post
```

Este código crea un archivo Post. php dentro de la carpeta App/Models.
Crea un array <fillable. Añade el siguiente código dentro de la clase Post y debajo de la línea use HasFactory;

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Post extends Model
{
    use HasFactory;

    protected $fillable = [
        'title',
        'body',
    ];
}
```

Este código crea un array fillable que te permite añadir elementos a la base de datos desde tu aplicación Laravel.

Conecta el modelo Post al archivo PostController.php. Abre PostController.php y añade la siguiente línea en use Illuminate\Http\Request;. Tiene el siguiente aspecto(Lo que mencione arriba):

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\Post;

class PostController extends Controller
{
```

Después de crear las funciones del controlador y el modelo Post, debes añadir rutas para las funciones de tu controlador.

Abre routes/web.php y elimina la ruta boilerplate que generó la aplicación. Sustitúyela por el código que aparece a continuación para conectar las funciones del controlador a sus respectivas rutas:
Cambiamos `URL::forceScheme('https');` a `URL::forceScheme('http');` sino nos falla.

```
<?php

use Illuminate\Support\Facades\Route;
use App\Http\Controllers\PostController;
use Illuminate\Support\Facades\URL;

/*
|--------------------------------------------------------------------------
---
| Web Routes
|--------------------------------------------------------------------------
---
|
| Here is where you can register web routes for your application. These
| routes are loaded by the RouteServiceProvider within a group which
| contains the "web" middleware group. Now create something great!
|
*/
//URL::forceScheme('https');
URL::forceScheme('http');
// returns the home page with all posts
Route::get('/', PostController::class .'@index')->name('posts.index');
// returns the form for adding a post
Route::get('/posts/create', PostController::class .
'@create')->name('posts.create');
// adds a post to the database
Route::post('/posts', PostController::class
.'@store')->name('posts.store');
// returns a page that shows a full post
Route::get('/posts/{post}', PostController::class
.'@show')->name('posts.show');
// returns the form for editing a post
Route::get('/posts/{post}/edit', PostController::class
.'@edit')->name('posts.edit');
// updates a post
Route::put('/posts/{post}', PostController::class
.'@update')->name('posts.update');
// deletes a post
Route::delete('/posts/{post}', PostController::class
```

```
.'@destroy')->name('posts.destroy');
```

Para conectar las rutas, abre web.php y añade la línea siguiente debajo de la línea **use Illuminate\Support\Facades\Route;** En realidad esto ya esta hecho en el código de arriba, mal explicado por el tutorial, uno agrega **use App\Http\Controller\PostController;** Como puede observarse en la cuarta línea:

```
<?php

use Illuminate\Support\Facades\Route;
use App\Http\Controllers\PostController;
use Illuminate\Support\Facades\URL;
```

Pero es algo que nosotros deberíamos agregar manualmente.

GENERAR vistas BLADE:
Ahora que tienes las rutas, puedes crear los archivos Blade de Laravel. Antes de utilizar Artisan para generar los archivos Blade, crea el comando make:view, con el que podrás generar los archivos blade.php.

Ejecuta el siguiente código en la CLI para crear un archivo de comando MakeViewCommand dentro de la carpeta app/Console/Commands :

```
php artisan make:command MakeViewCommand
```

Crea un comando para generar archivos .blade.php desde la CLI sustituyendo el código del archivo MakeViewCommand por lo siguiente:

```
<?php

namespace App\Console\Commands;

use Illuminate\Console\Command;
use File;

class MakeViewCommand extends Command
{
    /**
     * The name and signature of the console command.
     *
     * @var string
     */
    protected $signature = 'make:view {view}';
```

```php
    /**
     * The console command description.
     *
     * @var string
     */
    protected $description = 'Create a new blade template.';

    /**
     * Execute the console command.
     *
     * @return mixed
     */
    public function handle()
    {
        $view = $this->argument('view');

        $path = $this->viewPath($view);

        $this->createDir($path);

        if (File::exists($path))
        {
            $this->error("File {$path} already exists!");
            return;
        }

        File::put($path, $path);

        $this->info("File {$path} created.");
    }

     /**
     * Get the view full path.
     *
     * @param string $view
     *
     * @return string
     */
    public function viewPath($view)
    {
        $view = str_replace('.', '/', $view) . '.blade.php';

        $path = "resources/views/{$view}";

        return $path;
    }
```

```
    /**
     * Create view directory if not exists.
     *
     * @param $path
     */
    public function createDir($path)
    {
        $dir = dirname($path);

        if (!file_exists($dir))
        {
            mkdir($dir, 0777, true);
        }
    }
}
```

Me dio error con undefined File pero funciono!

Esto es totalmente extra! Otra cosa no enseñada, como artisan carece de un comando para crear vistas, todos parecen hacer esto para no estar creando archivo por archivo manualmente, se inventan un comando y lo reutilizan.

Entonces gracias a este comando creado podemos hacer lo siguiente:

Crear una Página de Inicio

A continuación, crea tu página de inicio. La página de inicio es el archivo index.blade.php, que enumera todas las entradas.

Para crear la página de inicio, ejecuta:

```
php artisan make:view posts.index
```

Esto crea una carpeta posts dentro de la carpeta /resources/views y un archivo index.blade.php debajo de ella. La ruta resultante es /resources/views/posts/index.blade.php.

Añade el siguiente código dentro del archivo index.blade.php:

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/css/boots
```

```
trap.min.css" rel="stylesheet"

integrity="sha384-GLhlTQ8iRABdZLl6O3oVMWSktQOp6b7In1Zl3/Jr59b6EGGoI1aFkw
7cmDA6j6gD" crossorigin="anonymous">

    <title>CRUDPosts</title>
</head>

<body>
    <nav class="navbar navbar-expand-lg navbar-light bg-warning">
        <div class="container-fluid">
            <a class="navbar-brand h1" href={{ route('posts.index')
}}>CRUDPosts</a>
            <div class="justify-end ">
                <div class="col ">
                    <a class="btn btn-sm btn-success" href={{
route('posts.create') }}>Add Post</a>
                </div>
            </div>
    </nav>
    <div class="container mt-5">
        <div class="row">
            @foreach ($posts as $post)
                <div class="col-sm">
                    <div class="card">
                        <div class="card-header">
                            <h5 class="card-title">{{ $post->title
}}</h5>
                        </div>
                        <div class="card-body">
                            <p class="card-text">{{ $post->body }}</p>
                        </div>
                        <div class="card-footer">
                            <div class="row">
                                <div class="col-sm">
                                    <a href="{{ route('posts.edit',
$post->id) }}"
                                        class="btn btn-primary
btn-sm">Edit</a>
                                </div>
                                <div class="col-sm">
                                    <form action="{{
route('posts.destroy', $post->id) }}" method="post">
                                        @csrf
                                        @method('DELETE')
                                        <button type="submit" class="btn
```

```
btn-danger btn-sm">Delete</button>
                                        </form>
                                    </div>
                                </div>
                            </div>
                        </div>
                    @endforeach
                </div>
            </div>
        </body>

        </html>
```

El código anterior crea una página HTML sencilla que utiliza Bootstrap para el estilo. Establece una barra de navegación y una plantilla de cuadrícula que enumera todas las entradas de la base de datos con detalles y dos botones de acción — editar y eliminar — utilizando el ayudante @foreach Blade.

El botón Edit lleva al usuario a la página Edit post, donde puede editar la entrada. El botón Delete borra la entrada de la base de datos utilizando {{ route('posts.destroy', $post->id) }} con un método DELETE.

Nota: El código de la barra de navegación de todos los archivos es el mismo que el del archivo anterior.

Crea la página create.blade.php. El archivo Blade llamado create añade entradas a la base de datos. Utiliza el siguiente comando para generar el archivo:

```
php artisan make:view posts.create
```

Esto genera un archivo create.blade.php dentro de la carpeta /resources/views/posts .

Añade el siguiente código al archivo create.blade.php:

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/css/boots
trap.min.css" rel="stylesheet"

integrity="sha384-GLhlTQ8iRABdZLl6O3oVMWSktQOp6b7In1Zl3/Jr59b6EGGoI1aFkw
```

```
7cmDA6j6gD" crossorigin="anonymous">

    <title>Create Post</title>
</head>

<body>
    <nav class="navbar navbar-expand-lg navbar-light bg-warning">
        <div class="container-fluid">
            <a class="navbar-brand h1" href={{ route('posts.index')
}}>CRUDPosts</a>
            <div class="justify-end ">
                <div class="col ">
                    <a class="btn btn-sm btn-success" href={{
route('posts.create') }}>Add Post</a>
                </div>
            </div>
    </nav>

    <div class="container h-100 mt-5">
        <div class="row h-100 justify-content-center
align-items-center">
            <div class="col-10 col-md-8 col-lg-6">
                <h3>Add a Post</h3>
                <form action="{{ route('posts.store') }}" method="post">
                    @csrf
                    <div class="form-group">
                        <label for="title">Title</label>
                        <input type="text" class="form-control"
id="title" name="title" required>
                    </div>
                    <div class="form-group">
                        <label for="body">Body</label>
                        <textarea class="form-control" id="body"
name="body" rows="3" required></textarea>
                    </div>
                    <br>
                    <button type="submit" class="btn btn-primary">Create
Post</button>
                </form>
            </div>
        </div>
    </div>
</body>

</html>
```

El código anterior crea un formulario con los campos title y body y un botón submit para añadir una entrada a la base de datos a través de la acción {{ route('posts.store') }} con un método POST.

Crea la página Edit post para editar entradas en la base de datos. Utiliza el siguiente comando para generar el archivo:

```
php artisan make:view posts.edit
```

Esto crea un archivo edit.blade.php dentro de la carpeta /resources/views/posts .

Añade el siguiente código al archivo edit.blade.php:

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/css/boots
trap.min.css" rel="stylesheet"

integrity="sha384-GLhlTQ8iRABdZLl6O3oVMWSktQOp6b7In1Zl3/Jr59b6EGGoI1aFkw
7cmDA6j6gD" crossorigin="anonymous">

    <title>Edit Post</title>
</head>

<body>
    <nav class="navbar navbar-expand-lg navbar-light bg-warning">
        <div class="container-fluid">
            <a class="navbar-brand h1" href={{ route('posts.index')
}}>CRUDPosts</a>
            <div class="justify-end ">
                <div class="col ">
                    <a class="btn btn-sm btn-success" href={{
route('posts.create') }}>Add Post</a>
                </div>
            </div>
    </nav>
    <div class="container h-100 mt-5">
        <div class="row h-100 justify-content-center
align-items-center">
            <div class="col-10 col-md-8 col-lg-6">
                <h3>Update Post</h3>
```

```
                <form action="{{ route('posts.update', $post->id) }}"
method="post">
                    @csrf
                    @method('PUT')
                    <div class="form-group">
                        <label for="title">Title</label>
                        <input type="text" class="form-control"
id="title" name="title"
                               value="{{ $post->title }}" required>
                    </div>
                    <div class="form-group">
                        <label for="body">Body</label>
                        <textarea class="form-control" id="body"
name="body" rows="3" required>{{ $post->body }}</textarea>
                    </div>
                    <button type="submit" class="btn btn-primary">Update
Post</button>
                </form>
            </div>
        </div>
    </div>
</body>

</html>
```

El código anterior crea un formulario con los campos title y body y un botón de envío para editar una entrada con el id especificado en la base de datos a través de la acción {{ route('posts.update') }} con un método PUT.

A continuación, reinicia tu servidor de aplicaciones utilizando el código siguiente:

```
php artisan serve
```

Con lo cual no usa apache
Visita **http://127.0.0.1:8000** en tu navegador para ver tu nuevo blog. Haz clic en el botón Add post para añadir nuevas entradas.

Solo muestra index, el resto ni mu, el error es porque el tutorial es de un hosting que como es normal usa https y nosotros localmente usamos http, así que hay que cambiar eso.
Funcionó tras cambiar en routes/web.php la línea:

```
//URL::forceScheme('https'); Nos causa falla
URL::forceScheme('http');
```

Basado en “mi resumen” afanado de Cómo hacer CRUD (Create, Read, Update, and Delete. Crear, Leer, Actualizar y Eliminar) con Laravel (kinsta.com) y como tiene errores de redacción y ausencias de código, aca su github.