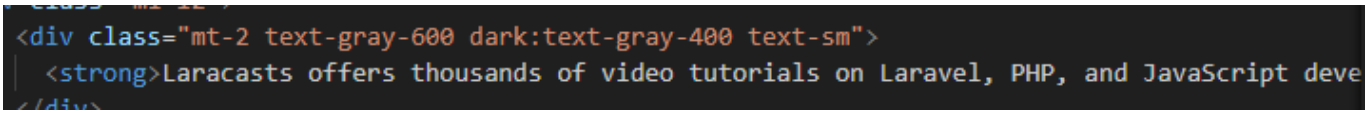
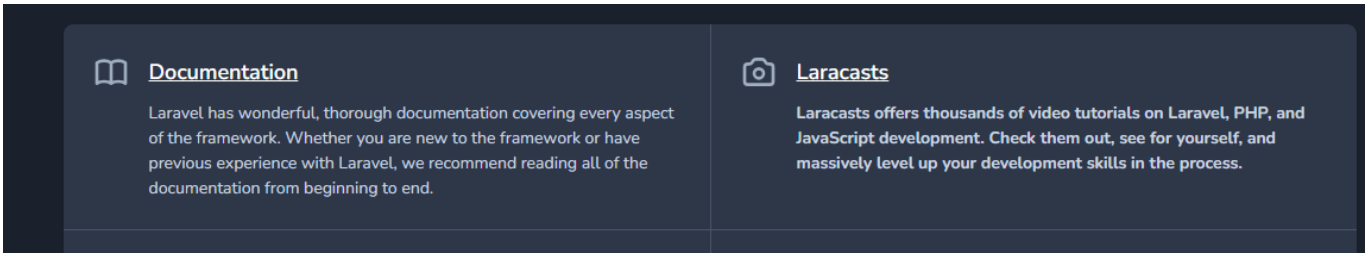
# LFTS - The Basis

## 05. How a Route Loads a View

### Como funcionan las rutas

En la primera parte nos muesta como fuciona las rutas y nos muestra donde esta la pagina por default de laravel, en esta pagina y a modo de demostracion cambiamos un poco esta pagina encerrando uno de los textos en una etiquetas "strong" y vemos como cambia en la pagina



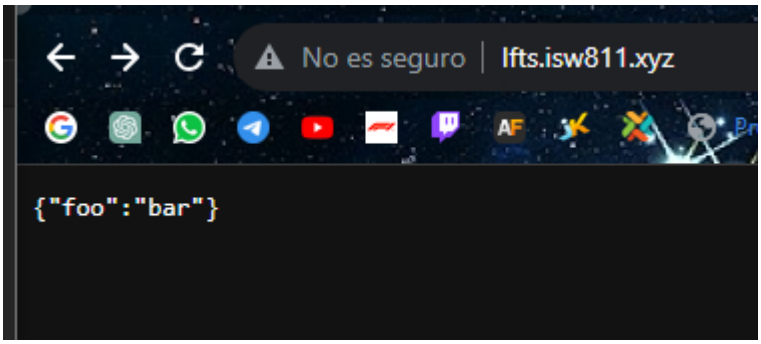




Aparte de vistas en las rutas podemos imprimir texto plano

```php
Route::get('/', function () {
    return ('hello world');
});
```


Aparte podemos incluso retornar Json

```php
Route::get('/', function () {
    return ['foo' => 'bar'];
});
```




## 06. Include CSS and JavaScript

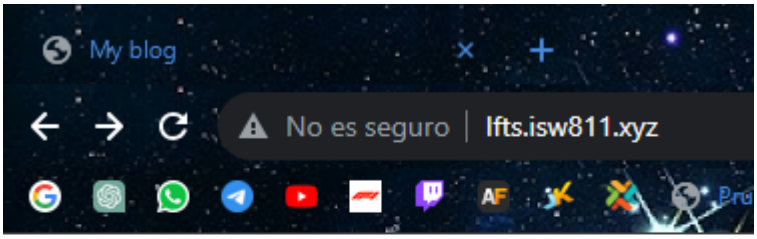
Empezamos modificando por completo la pagina por defecto de laravel y introduciomos el siguiente codigo

resources > views > welcome.blade.php

```
<title> My blog </title>

<body>
    <h1> Hello World</h1>
</body>
```

Y obiamente el resultado va a ser el siguiente




# Hello World

## Implementacion CSS

Vamos a la public y creamos una carpeta que se va a llamar "app.css" y le metemos una modificaciones para la pagina

public > # app.css > body

```
body {
background: navy;
color: white;
}
```

Y obiamente el resultado va a ser el siguiente


creamos un archivo JS que como resultado da lo siguiente


## 07. Make a Route and Link to it

Modificamos el nombre del archivo por defecto para que pase de "Welcome" a "posts" y le agregamos unas pequeñas modificaciones al la pagina que veniamos creando, tambien modificamos el CCS para que sea un texto decente de utilizar

posts.blade.php

```
eb.php M    posts.blade.php U ×    # app.css M
urces > views > posts.blade.php
<title> My blog </title>
<link rel="stylesheet" href="/app.css">
<body>
    <article>
    <h1><a href="/post"></a> My first post</h1>

    <p>Lorem ipsum, dolor sit amet consectetur adipisicing elit. Culpa perferendis ipsum mollitia,
        voluptatem error assumenda enim. Aspernatur autem perspiciatis provident possimus,
        quam cupiditate sequi recusandae quae distinctio dolorum impedit cum.</p>
    </article>

    <article>
    <h1><a href="/post"></a> My Second po st</h1>

    <p>Lorem ipsum, dolor sit amet consectetur adipisicing elit. Culpa perferendis ipsum mollitia,
        voluptatem error assumenda enim. Aspernatur autem perspiciatis provident possimus,
        quam cupiditate sequi recusandae quae distinctio dolorum impedit cum.</p>
    </article>

    <article>
    <h1><a href="/post"></a> My Third post</h1>

    <p>Lorem ipsum, dolor sit amet consectetur adipisicing elit. Culpa perferendis ipsum mollitia,
        voluptatem error assumenda enim. Aspernatur autem perspiciatis provident possimus,
        quam cupiditate sequi recusandae quae distinctio dolorum impedit cum.</p>
    </article>
</body>
```

Ahora vamos a crear un nuevo archivo que se va a llamar "post" para la ruta que vamos a crear y la cual la va a redirigir los titulos de los post que creamos en posts

```
Route::get('post', function (){
    return view('post');
```


## 08. Store Blog Posts as HTML Files

Ahora basicamente ocupamos que los post que estamos creando sean dinamicos y no tenga problemas para eso podemos asigarle una variable a post en vez del texto quemado, pero ocupamos configurar la rutas para esto

## creacion de la variable en post

Vamos a crear una ruta dinamica para cada post

```
<body>
    <article>
        <?= $post; ?>
    </article>

    <a href="/">Go Back</a>
```

## configuracion de la ruta dinamica

Al crear el codigo necesario para tener acesso de manera dinamica a los archivo tambien vamos a crear un control de errores, com se muestra en el codigo a continuacion

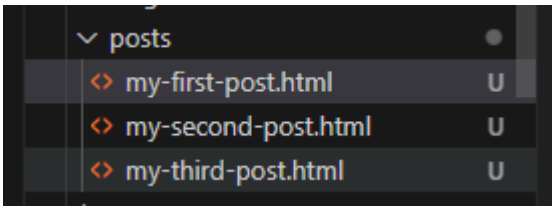
```
Route::get('posts/{post}', function ($slug){
    $path = __DIR__ . "/../resources/posts/{$slug}.html";
    if(! file_exists($path)){
        // abort(404);
        return redirect('/'); // redirecciona a la pagina principal
    }

    $post = file_get_contents($path);

    return view('post', [
        'post' => $post
    ]);
});
```

## creacion de los archivos

Para estos archivos vamos a crearlos en la carpetas de resources, creamos una carpeta posts y alli metemos  los archivos HTML de los post




Resultado

> Ahora podemos acceder a cada uno de los archivos post de una manera un poco mas dinamica


## 09. Route Wildcard Constraints

> En este episodio del curso basicamente le agregamos un poco de seguridad a la ruta del wildcart para que solo accepte un grupo que nosotros asignamos atravez de regular expressions

```php
Route::get('posts/{post}', function ($slug){
    $path = __DIR__ . "/../resources/posts/{$slu
    if(! file_exists($path)){
        // abort(404);
        return redirect('/'); // redirecciona a
    }

    $post = file_get_contents($path);

    return view('post', [
        'post' => $post
    ]);
})-> where('post', '[A-z_\-]+');
```

## 10. Use Caching for Expensive Operations

En este episodio nos enceñan a cachear cierta informacion en ese caso en especifico el "$slug" para evitar gastar recursos o recargar informacion innecesariamente, tambien acamodamos un poco el codigo

```php
Route::get('posts/{post}', function ($slug){

    if(! file_exists($path = __DIR__ . "/../resources/posts/{$slug}.html")){
        return redirect('/'); // redirecciona a la pagina principal
    };

   $post = cache()->remember("posts.($slug", 1200, fn() => file_get_contents($path));

    return view('post', ['post' => $post]);
})-> where('post', '[A-z_\-]+');
```

## 11. Use the Filesystem Class to Read a Directory

Ahora vamos a crear una manera en la cual no importa si agregamos mas post siempre sea visible, para esto vamos a necesitar crear una model para los post en el cual seremos podremos cargar los post y obiviamente para que esto funcione abra que hacer modificaciones al php de rutas y al posts.blade.php

Hacemos un foreach para cargar los post

```php
<body>
    <?php foreach($posts as $post) : ?>
    <article>
        <?= $post; ?>
    </article>
    <?php endforeach; ?>
</body>
```

Modificacion de el archivo de rutas "web" para que funcionen nuestras modificaciones

Aqui vemos 2 rutas una basicamente es la ruta de la pagina principal que carga todos los post y la segunda que esta destina para cargar un post en especifico, el que nosostros especifiquemos clickeando el link

```php
Route::get('/', function () {
    return view('posts', [
        'posts' => Post::all()
    ]);
});

Route::get('posts/{post}', function ($slug){
    return view('post',[
        'post'=> Post::find($slug)
]);
})-> where('post', '[A-z_\-]+');
```

### Creacion del Modelo Post

```
Aqui creamos las funciones que van a hacer la "logica" para cargar los posts
```

```php
class Post
{
public static function all()
{
    $files = File::files(resource_path("posts/"));

    return array_map(fn($file) => $file->getContents(), $files);
}

public static function find($slug)
    {
        if(!file_exists($path = resource_path("posts/{$slug}.html"))){
            throw new ModelNotFoundExeption();
        }

        return $post = cache()->remember("posts.($slug", 1200, fn() => file_get_contents($path));

    }
```

## 12. Find a Composer Package for Post Metadata

para este episodio ocupamos instalar una nueva depencencia con composer para esto tenemos que ir a la maquina virtual y ejecutar el siguiente codigo

```
cd sites/lfts.isw811.xyz
composer require spatie/yaml-front-matter
```

ahora con esto instalado tenemos lo ultimo que ocupamos para hacer el programa 100% dinamico

### Modelo Post

En el modelo tenemos que hacer un constructor y rehacer el codigo de las funciones para coger todos los post y la funcion que se utiliza para cargar uno en especifico, en este caso nos podemos aprovechar de la funcion de cargar todos los post para cargar uno en especifico utilizando el slug que metimos en la metadata de cada post

```php
public function __construct($title, $excerpt, $date, $body, $slug)
{
    $this->title = $title;
    $this->excerpt = $excerpt;
    $this->date = $date;
    $this->body = $body;
    $this->slug = $slug;
}

public static function all()
{
   return collect(File::files(resource_path("posts")))
    ->map(fn($file) => YamlFrontMatter::parseFile($file))
    ->map(fn ($document) => new Post(
            $document-> title ,
            $document-> excerpt,
            $document-> date,
            $document-> body(),
            $document-> slug,
        ));
}

public static function find($slug)
    {
       return static::all()->firstWhere('slug',$slug);
    }
```

Routes web

```php
Route::get('/', function () {
    $posts = Post::all();

    return view('posts', [
        'posts' => $posts
    ]);
});

Route::get('posts/{post}', function ($slug){
    return view('post',[
        'post'=> Post::find($slug)
]);
})-> where('post', '[A-z_\-]+');
```

Post.blade.php

```
<article>
<h1><?= $post->title; ?> </h1>

<div>
    <?= $post->body; ?>
</div>
</article>
```

Posts.blade.php

```
<body>
    <?php foreach($posts as $post) : ?>
    <article>
        <h1>
            <a href="/posts/<?= $post->slug; ?>">
                <?= $post->title; ?>
            </a>

        </h1>

        <div>
        <?= $post->excerpt; ?>
        </div>

    </article>
    <?php endforeach; ?>
```

## 13. Collection Sorting and Caching Refresher

> Ahora acomodamos por fecha los posts que tenemos y tambien cachemos el los posts para siempre y lo revisamos con el artisan tinker

```
    return cache()-> rememberForever('posts.all', function() {
    return collect(File::files(resource_path("posts")))
        ->map(fn($file) => YamlFrontMatter::parseFile($file))
        ->map(fn ($document) => new Post(
            $document-> title ,
            $document-> excerpt,
            $document-> date,
            $document-> body(),
            $document-> slug,
        ))
        -> sortByDesc('date');
    });
```

Artisan thinker

En la maquina virtual abrimos el artisan thiker y reviamos la informacion cacheada

```
cd /sites/lfts.isw811.xyz
php artisan thinker
cache()->get(posts.all);
```