Taller de Laravel 5.1 LTS

El framework que revolucionó PHP

Contenido

Material

- Github
- Estos slides
- Laravel
- Laracasts

Sesión 1: Basics

1. ¿Qué es Laravel?
2. Instalación
   a. Composer
   b. Laravel installer
3. Configuración
4. Funciones anónimas (closures)
5. Rutas
6. Vistas
7. Paso de variables
8. Blade

Actividad

- Rutas y vistas para una landing page

Material

- Github

¿Qué es un framework?

Estructura conceptual y tecnológica de soporte definido, normalmente con **artefactos o módulos** concretos de software, que puede servir de base para la organización y desarrollo de software.

¿Qué es Laravel?

Framework de código abierto para desarrollar aplicaciones y servicios web con PHP 5. Su filosofía es desarrollar código PHP de forma elegante y simple, evitando el "código espagueti". Fue creado en 2011 por Taylor Otwell.

contacto@corb.mx
+52 33 3333-0351

Bernardo de Balbuena 307
Col. Ladrón de Guevara, Guadalajara, Jal.

Características

- Sistema de ruteo, también RESTful
- Blade, Motor de plantillas
- Eloquent ORM
- Uso de Composer
- Soporte para el caché
- Soporte para MVC
- Usa componentes de Symfony
- Adopta las especificaciones PSR-2 y PSR-4

Instalación

Requisitos:

- PHP >= 5.5.9
- OpenSSL PHP Extension
- PDO PHP Extension
- Mbstring PHP Extension
- Tokenizer PHP Extension

¿Qué es composer?

- Composer es un manejador de dependencias, no un gestor de paquetes.
- Trata con paquetes y bibliotecas
- La instalación siempre es local para cualquier proyecto, las bibliotecas se instalan en un directorio por defecto (normalmente es /vendor).
- Composer es capaz de instalar las bibliotecas que requiere tu proyecto con las versiones que necesites.
- ¿Y si mis bibliotecas dependen de otras? También es capaz de resolver eso y descargar todo lo necesario para que funcione.

Instalar con composer

Para instalar laravel con composer se ejecuta el comando

```
$ composer create-project laravel/laravel taller "5.1.*"
```

Para levantar el servidor, dentro del folder del proyecto

```
$ php artisan serve
```

Instalar con Laravel Installer

Descargar el instalador de laravel

```
$ composer global require "laravel/installer"
```

Agregar laravel al path

```
$ export PATH="$PATH:$HOME/.composer/vendor/bin"
```

Instalación limpia de Laravel en la versión mas reciente

```
$ laravel new taller
```

Configuración

- Permisos de escritura al web server para **storage** y **bootstrap/cache**
- Setear key `$ php artisan key:generate`
- Zona horaria y otros detalles en **config/app.php** pero practicamente todo lo toma de **.env**
- Configuración de base de datos **.env** y otras configuraciones básicas

Arquitectura del proyecto

```
app/
    Http/
        routes.php
resources/
    views/
```

Arquitectura del proyecto

app/ donde se encuentran las clases de modelos, controladores, rutas

bootstrap/ contiene configuraciones del framework y cache

config/ donde se ubican los archivos de configuración del proyecto.

database/ donde creamos las migraciones, seeders y model factories, es decir, lo relacionado a la base de datos.

public/ contendrá todos los archivos que estarán disponibles para los usuarios.

resources/ se encuentran los assets de nuestra aplicación

storage/ archivos temporales del framework.

tests/ para ubicar todas las pruebas que tendrá la aplicación.

vendor/ donde se alojaran los componentes de terceros manejados por Composer.

Closures

Funciones que no tienen un nombre específico

```
Route::get('/', function () {
    return view('welcome');
});
```

Rutas

Respuestas de HTTP

- get
- post
- put
- patch
- delete
- options

```
Route::get('/', function () {
    return view('welcome');
});
```

Vistas

- Estan en resources/views
- El archivo debe ser un nombre.blade.php
- Pueden estar dentro de un subfolder
- Para llamar vistas en las rutas que estan dentro de un subfolder se usa punto, ejemplo **folder.vista**

Paso de variables

- Pasar un arreglo creado en el momento

```
return view('welcome', ['var' => $var]);
```

- Utilizar un arreglo que esta en la codificación

```
return view('welcome', compact('var'));
```

- Con método nativo de Laravel

```
return view('welcome')->with('var', $var);
```

- Aún mas “elegante” *donde “Var” es el nombre que se le quiere dar a la variable*

```
return view('welcome')->withVar($var);
```

Bloques en Blade

Directivas

- @if / @else / @endif
- @unless / @endunless
- @foreach / @endforeach
- @forelse / @empty / @endforelse

Interpolación

- {{ $var }}
- {{ algunaFuncion() }}

Repaso de Sesión 1

contacto@corb.mx
+52 33 3333-0351

Bernardo de Balbuena 307
Col. Ladrón de Guevara, Guadalajara, Jal.

¿Qué es un framework?

Un conjunto de funcionalidades para facilitar el desarrollo

¿Qué es Laravel?

Un framework de PHP, basado en 5, creado en 2011, con MVC, busca la elegancia y simplicidad

¿Qué significa LTS en la versión?

Long Term Support

¿Qué es composer?

Un manejador de dependencias para PHP

¿Cómo instalar Laravel?

composer create-project laravel/laravel

¿En donde se guarda la configuración de mi proyecto?

.env

¿Dónde se encuentra el archivo de rutas?

app/Http/routes.php

¿Dónde se encuentran las vistas?

resources/views

¿Qué es artisan?

la consola de laravel

¿Qué es un closure?

una función anónima

¿Cómo se pasan valores a una vista?

array directamente
compact
with
withKey

¿Qué es blade?

Manejador de plantillas

¿Cuáles son los directivas de blade para bloques de sentencias?

con @ y la sentencia

¿Cómo se hace la interpolación en blade?

con llaves

{{ $var }}

Sesión 2: Controladores

1. Rutas dinámicas
   a. Parámetros
   b. Parámetros opcionales
   c. Validación de parámetros
2. Helpers
   a. dd
3. Modo debug
4. MVC
5. Controladores
   a. archivo a patita
   b. con artisan

Actividad:

- Rutas RESTful para Todo List
- Controlador y sus métodos para Todo List

Material

- Github
- http://restcookbook.com/

Parámetros en las rutas

Cuándo deseamos usar rutas con datos dinámicos

```
Route::get('{id}' , function () {} );
```

Validación de los argumentos con **where**

```
Route::get('{id}', function () {
    ...
})->where('id','[0-9]+');
```

```
->where(['id' => '[0-9]+', 'name' => '[a-z]+']);
```

Obtener los parámetros de las rutas

Cuándo deseamos usar los parámetros de las rutas se deben pasar como argumentos en la función

```
Route::get('{id}' , function ($id) {
    return $id;
});
```

Helpers

Funciones para ayudar en las funcionalidades típicas y no tan triviales

dd() dump de la variable y exit

Activar y desactivar el modo debug

1. En el archivo **.env** cambiar a true o false el APP_DEBUG
2. Reiniciar el servidor

Modelo · Vista · Controlador

Cargar controladores desde las rutas

Que la ruta mande a llamar un controlador

```
Route::get('{id}', 'MyController');
```

Que se ejecute el método del controlador con **@método**

```
Route::get('/{id}', 'MyController@inicio');
```

Crear un controlador

app/Http/Controllers/archivo.php

```
namespace App\Http\Controllers;

class MyController extends Controller
{

}
```

Con artisan

Crear el controlador con artisan a partir de 5.1 lo va a generar como un recurso con todos sus métodos para RESTful

```
php artisan make:controller nombre-controlador
```

Funcionalidad en los controladores

Toda la funcionalidad que al momento teníamos en nuestra ruta debería estar en nuestro controlador

```
Route::get('{id}' , 'MyController@index');
```

```
class MyController extends Controller
{
    public function index($id)
    {
        return $id;
    }
```

Repaso de Sesión 2

contacto@corb.mx
+52 33 3333-0351

Bernardo de Balbuena 307
Col. Ladrón de Guevara, Guadalajara, Jal.

¿Cómo se definen rutas dinámicas?

entre llaves
Route::get(‘ruta/{dinamico}’,

¿Cómo se define que el parámetro de una ruta es opcional?

con interrogación

Route::get(‘ruta/{opcional?}’

¿Cómo se validan los parámetros de una ruta?

con where y expresión regular
->where('valor','regex')

Menciona dos helpers

dd() - dump y die

¿Cómo se activa o desactiva el modo debug?

En el archivo .env la constante APP_DEBUG y se reinicia el servidor

¿Qué es MVC?

Patrón de diseño
Modelo Vista Controlador

¿Cuál es la funcionalidad de un controlador?

Validación de request, autenticación, controlar la ejecución

¿Cuál es la funcionalidad de un modelo?

Lógica del negocio, ORM, base de datos

¿Cuál es la funcionalidad de una vista?

Mostrar infomación

¿En donde se guardan los controladores?

app/Http/Controllers

¿Cuál es la estructura del archivo de un controlador?

Namespace

Use de paqueterías

Clase que extiende de Controller

Métodos

¿Cómo se crean controladores con artisan?

```
php artisan make:controller nombre
```

¿Qué incluye un controlador creado con artisan?

Todos los métodos de un resource

¿Qué métodos HTTP soporta laravel en sus rutas?

Get, Post, Put, Patch, Delete, Options

Sesión 3: Modelos

1. Estándar para peticiones RESTful
2. Migrations
3. Tinker
4. Query Builder
5. Modelos (ORM)
6. Eloquent
7. Rutas con modelos

Actividad:

- RESTful con base de datos del ToDo list

Material

- Github
- Migrations

Peticiones RESTful

En Laravel se entiende el concepto de “recursos” y para estos la estructura indicada es:

¿Qué queremos hacer?	Método HTTP	Ruta	Método del controller
Listar todos los recursos	get	recursos	index
Mostrar formulario para crear	get	recursos/create	create
Crear un recurso	post	recursos	store
Mostrar un recurso	get	recursos/{id}	show
Mostrar formulario para editar	get	recursos/{id}/edit	edit
Editar un recurso	put	recursos/{id}	update
Eliminar recurso	delete	recursos/{id}	destroy

Migrations

- Archivos en los cuales se define la estructura de la base de datos de manera que se puede versionar para trabajo en equipo.
- Con opciones para la creación de la tabla pero también para “deshacer” los cambios en caso que se requiera.

Crear migrate con artisan

Como todo lo ya hecho hasta el momento, con artisan podemos crear nuestros migrates y luego completarlos con más información.

```
php artisan make:migration nombre_del_migrate [options]
```

[options]

--create tabla_a_crear

--table tabla_a_modificar

Agregar campos

Los campos a agregar se tratan como elementos del objeto $table. Y existe una lista de métodos a utilizar de acuerdo al tipo de dato.

```
$table->string('campo');
```

Ejecutar los migrates

Laravel lleva control de los migrates que se han ejecutado, y para decirle que ejecute los migrates pendientes, entonces tenemos que pedirle con lo haga.

```
php artisan migrate
```

Modificar campos

A la columna que deseamos modificar en el migrate le aplicamos el change.

```
$table->string('name', 50)->change();
```

Nota: Requiere tener instalado doctrine

Eliminar campos

Para eliminar una columna con un migrate:

```
$table->dropColumn('votes');
```

Rollbacks

Deshacer el último migrate

```
php artisan migrate:rollback
```

Deshacer todos los migrates

```
php artisan migrate:reset
```

En 5.3 se puede deshacer hasta un migrate en específico con step y la cantidad de migrates hacia atrás.

Tinker

Shell interactivo **repl** (read-eval-print loop) para trabajar con la aplicación y la base de datos.

```
php artisan tinker
```

Query builder

- Seleccionar tabla

```
DB::table('nombre-tabla')
```

- Insertar elementos en una tabla

```
->insert([array de elementos])
```

```
DB::table('todos')->insert(['name' => 'mi todo', 'otro' =>
'1234']);
```

Query builder

- Obtener elementos

```
->get();
```

- Obtener el primer elemento

```
->first();
```

```
DB::table('todos')->get();
```

Query builder

- Condicional para igualdad

```
->where('campo', 'valor')
```

- Actualizar elementos en una tabla

```
->update([array de elementos])
```

```
DB::table('todos')->where('name' => 'mi todo');
```

Query builder

- Eliminar

```
->delete()
```

- Truncar una tabla

```
->truncate()
```

ORM

El mapeo objeto-relacional es una técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y la utilización de una base de datos relacional como motor de persistencia.

Eloquent

Es el ORM de Laravel y se llama “Eloquent” porque su sintaxis es elocuente y expresiva

Crear modelos con Eloquent

Crear un modelo

```
php artisan make:model nombre_de_clase [options]
```

[options]

-m Create también el migration.

Namespaces para Modelos

Recomiendo organizar los modelos por módulos o bloques y es necesario entonces ajustar los namespaces para que todo funcione como se debe.

```
namespace App\Models\Módulo;
```

Tablas y Modelos

- La tabla para la base de datos se usan en plural, el modelo, es decir, la clase se usa en singular.
- Se puede utilizar una tabla distinta asignado el valor en el atributo **$table**

```
protected $table = 'my_flights';
```

- La llave primaria se asume que es un campo de nombre id y que es un entero autoincrementable.

```
protected $primaryKey = 'id_vuelo';
```

- Existen los campos created_at y updated_at por default y se pueden desactivar poniendo el atributo **$timestamps** a false o cambiar el formato con **$dateFormat**.

Eloquent

- Obtener todos los elementos

```
all();
```

- Obtener un elemento por su id

```
find($id);
```

```
Modelo::all();
```

Ligando rutas con modelos

- Laravel internamente en la ruta puede ligar el parámetro con un modelo mientras se cree el binding.

```
Route::get('{thing}' , 'MyController@show');
Route::model('thing', '\App\Modelo');
```

- O poner todos los bindings en el router

```
public function boot(Router $router)
{
    $router->model('thing', 'App\Modelo');
}
```

Ligando rutas con modelos

- Y después solo inyectar el objeto en el método del controlador

```
public function show(Modelo $thing)
{
    return $thing;
}
```

Repaso de Sesión 3

contacto@corb.mx
+52 33 3333-0351

Bernardo de Balbuena 307
Col. Ladrón de Guevara, Guadalajara, Jal.

¿Qué son los migrations?

Clases que nos permiten manipular la base de datos

¿Cómo se crean los migrates?

```
php artisan make:migration nombre
```

¿Cómo se ejecutan los migrates?

```
php artisan migrate
```

¿Cómo se definen columnas en los migrates?

$table->tipodedato('campo')

¿Cómo se definen cambios a las columnas en los migrates?

```
$table->tipodedato('campo')
->change()
```

¿Qué es tinker?

shell para interactuar con laravel y la base de datos

¿Qué es el query builder?

Clase para manipular la base de datos

DB::

¿Cómo se selecciona una tabla?

DB::table('tabla')

¿Cómo se insertan elementos?

```
->insert([array])
```

¿Cómo se obtienen elementos?

->get()

¿Cómo se obtiene solo el primer registro?

->first()

¿Cómo agregan condicionales a la consulta?

```
->where('campo', 'valor')
```

¿Cómo se actualizan registros?

```
->update([array])
```

¿Qué es Eloquent?

ORM de laravel

¿Cómo se crean modelos?

```
php artisan make:model nombre
```

¿Cómo funcionan los namespaces?

como folders y se usan con use

Menciona algunas reglas de la relación entre tablas y módelos de Eloquent

Tabla en plural, modelo en singular

Primary key se llama id

Agrega los timestamps

¿Cómo obtengo todos los registros de un modelo?

```
Modelo::all();
```

¿Cómo se obtiene el registro por id?

```
Modelo::find($id);
```

Route Model Binding

Se define el router

Se inyecta el modelo

Sesión 4: Layouts y Formularios

1. Guardar con Eloquent
   a. Save
   b. Create
   c. Fillable y Guarded
2. Formularios
   a. Request
   b. CSRF
3. Redirects
4. Métodos put, patch y delete
5. Layouts
6. Gulp y Elixir

Actividad:

- Completar toda una aplicación de ToDos

Material

- Github
- Facades
- Elixir

Request

- Inyectandola en el método

```
public function store(Request $request)
$request->
```

- Con el facade

```
\Request::
```

- Con el helper

```
request()->
```

Guardar

1. Crear el objeto

```
$objeto = new \App\Modelo;
```

2. Asignarle los valores

```
$objeto->campo = 'valor';
```

3. Guardarlo

```
$objeto->save();
```

Crear

A la clase pedirle que cree un nuevo elemento y mandarle el arreglo con los datos

```
App\Modelo::create([array])
```

Podemos usar $request->all() para que nos traiga todos los datos de request en un arreglo

```
App\Modelo::create($request->all());
```

Fillable y Guarded

Los modelos de Eloquent están protegidos contra asignaciones masivas.

Metemos un excepción los campos que si queremos que sean editables con

```
protected $fillable = ['name'];
```

Y los que queremos que esten protegidos con

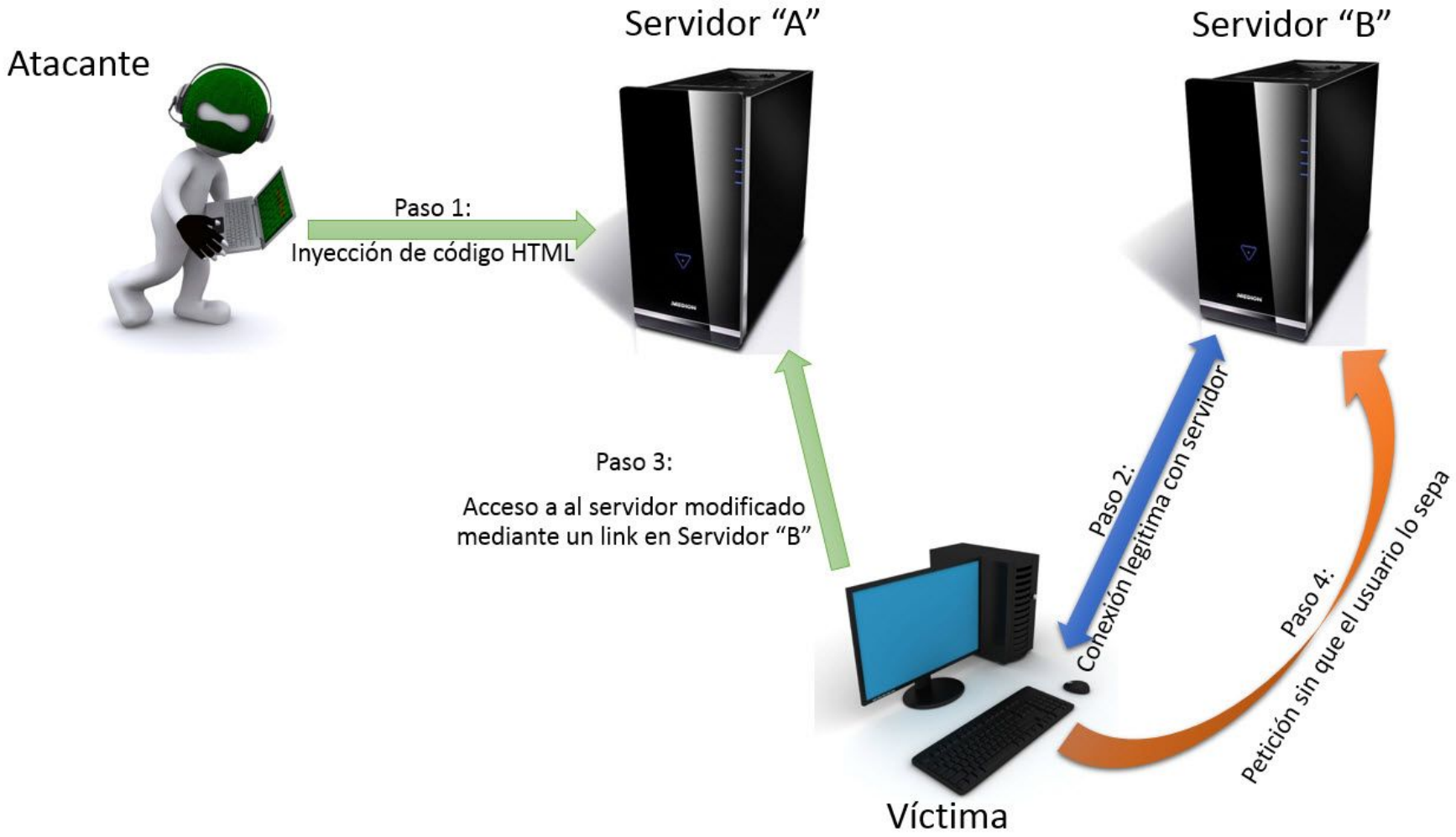
```
protected $guarded = ['id'];
```

Formularios

- Creamos un formulario con los campos necesarios
- Agregamos el método POST
- Agregamos la acción a la ruta que hayamos creado
- No olvidemos nuestro botón de submit

CSRF

Cross site request forgery (CSRF), también conocido como XSRF, es un ataque que consiste en ejecutar una acción desde el navegador comprometido a otra web, por lo cual estamos haciendo una petición “en nombre” de alguien mas.


Atacante
Servidor “A”
Servidor “B”
Paso 1:
Inyección de código HTML
Paso 2:
Conexión legitima con servidor
Paso 3:
Acceso a al servidor modificado
mediante un link en Servidor “B”
Paso 4:
Petición sin que el usuario lo sepa
Víctima

Protección CSRF

Laravel automáticamente genera un "token" CSRF para cada sesión de usuario activo manejado por la aplicación.

Este token es usado para verificar que el usuario autenticado es el que actualmente está haciendo la petición a la aplicación.

Token para CSRF

Agregamos un input que permitan mandar el token

```
<input type="hidden" name="_token" value="{{ csrf_token() }}">
```

O directo con blade

```
{{ csrf_field() }}
```

Desactivar CSRF

- En el archivo app/Http/Kernel.php comentar la linea del CSRF

\App\Http\Middleware\VerifyCsrfToken::class

- A agregar excepciones en el middleware

\App\Http\Middleware\VerifyCsrfToken en el arreglo $except

Redirects

Redigir a una nueva vista

Facade

`\Redirect::to(`

Helper

`redirect()->to(` o con la ruta directo `redirect('ruta')`

La vista previa

`back();`

Métodos que no son PUT y GET

Se simulan agregando un input hidden llamado _method

```
<input name="_method" type="hidden" value="PUT">
```

Pero mejor aún dejarselo a blade

```
{{ method_field('PUT') }}
```

Layouts

- Primero creamos un layout el cual contiene toda la arquitectura del sitio y define secciones.
- Luego por cada página indicamos que layout utilizar y creamos las secciones que vamos a utilizar.

yield

Insertar una sección de código en el layout

```
@yield(‘nombre-seccion')
```

extends

Utilizar un layout

```
@extends('nombre-layout')
```

Secciones en las vistas

En la vista, después de “incluir” el layout, se agregan las secciones que se desean

```
@section('nombre-seccion')
    Cosas aquí
@stop
```

Gulp

Gulp.js es un build system que permite automatizar tareas comunes de desarrollo como minificación de código JavaScript, recarga del navegador, compresión de imágenes, validación de sintaxis de código, etc.

Instalar Node.JS y entonces

```
npm install -g gulp
```

Elixir

Elixir es un API o wrapper para trabajar con Gulp y, de esta forma, poder automatizar tareas recurrentes en el desarrollo del front-end.

/gulpfile.js

Compilar sass y less

Podemos decir que compile nuestro SASS y automaticamente generará el archivo en public

```
elixir(function(mix) {
    mix.sass("app.scss");
});
```

Combinar archivos

```
elixir(function(mix) {
    mix.styles([
        "normalize.css",
        "main.css"
    ], 'public/build/css/everything.css');
});
```

Versionar archivos

```
elixir(function(mix) {
    mix.version("css/all.css");
});
```

Repaso de Sesión 4

contacto@corb.mx
+52 33 3333-0351

Bernardo de Balbuena 307
Col. Ladrón de Guevara, Guadalajara, Jal.

¿Cómo guardo un resource con Eloquent?

save()

create()

¿Por qué sale una excepción de Mass Assignment en algunas ocasiones?

Porque hay campos que no están definidos como que se tiene permisos para agregarlos

¿Cómo se soluciona la excepción Mass Assignment?

$fillable (lo que si)

$guarded (los que no)

¿Qué es un layout?

Machote, arquitectura, diseño base que se podrá reutilizar en muchas vistas

¿Cómo creo un layout?

1. Crear el archivo

¿Cómo uso un layout?

@extends('layout')

¿Cómo defino en un layout bloques dinámicos?

@yield('bloque')

¿En mi vista, como agrego el contenido dinámico para un bloque?

@section(‘bloque’)

...

@stop

¿Qué es CSRF?

Cross site request forgery

¿Cómo ayuda laravel para evitar CSRF?

Todas las peticiones deben enviar un token

¿Cómo creo un formulario en Laravel?

Un form común y corriente

Método definido

Action a la ruta tal como esta en routes.php

¿Cómo pongo el token para CSRF?

Input oculto con valor csrf_token()
y el name debe ser _token

o con blade {{ csrf_field() }}

¿Cómo finjo peticiones que no son get y post?

method_field(‘METODO’)

¿Qué son los Facade?

Clases estáticas para acceder a recursos

\Request::

¿Como hago un redirect en mi controlador?

\Redirect::to()

redirect()->to()

redirect(“)

back()

Sesión 5: Relaciones

1. Ver los queries generados por el query builder (tinker)
2. Algunos queries en general
3. Relaciones
4. Crear relación
   a. Llave foranea
   b. Método de la relación
5. Agregar y crear recursos relacionados

Actividad:

- Agregar comentarios a los ToDos

Material

- Github
-

Revisar los queries

```
DB::listen(function($query) { var_dump($query); })
```

Con esto podremos ver los queries que se crean en el tinker

Queries

```
Modelo::all();
Modelo::where('status',1)->get();
Modelo::where('status',1)->take(10)->get();
Modelo::where('thing','>=','a')->get()
```

Relaciones

Las relaciones se definen como métodos dentro del modelo con verbos muy claros como:

hasOne

belongsTo

hasMany

Pasos para crear relaciones

1. En la tabla que se va a recibir la relación, debe existir el campo a relacionar (llave foránea), entonces hay que crear el migrate.

```
$table->integer('foranea_id')->unsigned()->index();
$table->foreign('user_id')->references('id')->on('users');
```

2. Se debe crear un método para la relación en los modelos

```
public function relacion ()
```

3. Dentro del método de la relación entonces crear la relación con hasMany, belongsTo, etc

```
return $this->hasMany('\Modelo');
```

Usar recursos relacionados

- Crear elemento

```
$objeto = \Modelo::first();
```

- Asignarle un nuevo elemento

```
$objeto->relacion()->create($recurso);
```

- Obtener los elementos

```
$objeto->relacion();
```

Repaso de Sesión 5

contacto@corb.mx
+52 33 3333-0351

Bernardo de Balbuena 307
Col. Ladrón de Guevara, Guadalajara, Jal.

¿Cuándo se usa get()?

Obtener los resultados

¿Qué hace first()?

Obtener el primer elemento de la colección

¿Cómo se obtiene un solo campo en lugar del row completo?

value(‘campo’)

¿Cómo se eligen solo unos campos?

select(‘campo’,’campo2’...)

¿Cómo agrego condicionales a mi query builder?

```
where('campo','comparacion','valor'
)
```

¿Cómo mostrar el log generado con el query builder?

```
DB::listen(function($query) {
var_dump($query); })
```

Describe los pasos para crear relaciones

1. Migrate con campo para llave foranea

2. Método para la relación

3. En el método regresar segun el tipo de relacion ->hasMany()

Sesión 6: Usuarios y Auth

1. Usuarios
   a. Migrate
   b. bcrypt
2. Eagger Loading
3. Auth
   a. Rutas
   b. Vistas
   c. Controlador
   d. Usuarios
   e. Autenticación manual

Actividad:

- Los ToDos tienen usuarios
- Para acceder a los ToDos se requiere autenticación

Material

- Github
-

Creando usuarios

1. Hacer un migrate
2. Indicar las columnas que serán únicas `->unique();`
3. Password de mínimo 60 caracteres
4. Para trabajar con Auth de Laravel, agregar campo remember_token `$table->rememberToken();`

Cifrar contraseñas

Laravel permite mantener las contraseñas cifradas con su helper para aplicar un hash.

```
bcrypt('password');
```

Cargar los datos completos

Eagger loading es una manera de cargar toda la información cuando se requiere pero, además, en Laravel se busca reducir la cantidad de queries.

```
$objeto->load('relacion1.relacion dentro de 1');
```

```
$objeto->load('relacion1','relacion2');
```

Auth - Rutas

Laravel viene ya con toda la funcionalidad para manejar autenticación.

Paso número uno, crear las rutas

```
Route::get('auth/login', 'Auth\AuthController@getLogin');
Route::post('auth/login',
'Auth\AuthController@postLogin');
Route::get('auth/logout',
'Auth\AuthController@getLogout');
```

Auth - Vistas

Laravel ya tiene los controladores para manejo de autenticación, por lo que solo hace falta agregar las vistas.

Estas deben quedar en:

resources/views/auth/register.blade.php

resources/views/auth/login.blade.php

Auth - Controller

Los controladores ya están creados y solo debemos ajustarlos

app/Http/Controllers/Auth/AuthController.php

1. La ruta a la que se va a redirigir después de estar autenticado
   `protected $redirectPath =`
2. Ruta para cuando el usuario no está autenticado
   `protected $loginPath =`
3. Ajustes a la creación del usuario modificando el método **create**
4. Ruta pra cuando el usuario cierra su sesión
   `protectec redirectAfterLogout =`

El usuario

Datos del usuario

```
Auth::user();
```

```
$request->user()
```

Para revisar si el usuario está autenticado

```
Auth::check()
```

Middleware

Un middleware es un mecanismo que se utiliza para filtrar las peticiones HTTP en una aplicación.

El “intermediario” entre rutas y controladores.

Protección de rutas con middleware

Para proteger el acceso a una ruta utilizamos el middleware auth que ya trae laravel por defecto en app/Http/Middleware/Authenticate.php

```
Route::get('algos', [
    'middleware' => 'auth',
    'uses' => 'AlgoController@show'
]);
```

Grupos de rutas

Podemos agrupar distintas rutas para evitar en cada uno aplicar el mismo middleware por ejemplo

```
Route::group([
    'middleware' => 'auth',
    'prefix' => 'modulo',
], function () {
    Route::get('/', 'Controller@method');
    Route::get('user/profile', 'Controller@method');
});
```

Protección de rutas desde el controlador

En lugar de agregar el middleware en la ruta, lo podemos hacer en el constructor de nuestros controladores

```
public function __construct()
{
    $this->middleware('auth');
}
```

Autenticación manual

Login

```
Auth::attempt([datos])
```

Logout

```
Auth::logout();
```

Sesión 7: Middlewares y Validator

1. Sesiones
2. Flash
3. Middlewares
4. Manejo de Fechas
5. Validador
   a. Validate
   b. Request

Actividad:

- Un ToDo no puede cambiar de estatus en horario fuera de oficina
- Validación de formularios

Material

- Github
- Carbon
- Validator

Sesiones

Las sesiones por default están seteadas a archivos que es el estándar de PHP.

Si se desea cambiar la configuración se hace en el archivo config/session.php

Acceder a la sesión

\Session::

session()->

$request->session

Fluent

Session::put(‘asb’,’asdf’);

Session::get(‘asb’);

Session::has(‘asb’)

Flash

Notificar al usuario sobre una acción entre cambios de navegación, para eso usamos valores que solo existen para un request.

Creamos el mensaje en el controlador

```
session()->flash('mensaje','mensaje');
```

Mostramos el mensaje en nuestra vista

```
@if (session()->has('llave')):
    {{ session()->get('llave') }}
@endif
```

Middlewares

1. Crear un middleware
   ```
   php artisan make:middleware
   ```
2. Agregamos toda la lógica de validación antes de pasar al controlador
3. Return **next** si todo va bien, u otro return si la validación no pasó.
   ```
   return $next($request);
   ```

Trabajando con fechas

Carbon es una clase que se usa dentro de Laravel para facilitar el manejo de fechas.

Carbon\Carbon

- now() y today()
- diffForHumans()
- Operaciones add y sub
- Comparación con between

Validate

Dentro del controlador validamos el request o lo que queramos

```
$this->validate($request, [
        'title' => 'required|unique:posts|max:255',
        'body' => 'required',
    ]);
```

Accepted
Active URL
After (Date)
Alpha
Alpha Dash
Alpha Numeric
Array
Before (Date)
Between
Boolean
Confirmed
Date
Date Format
Different
Digits
Digits Between
E-Mail
Exists (Database)
Image (File)
In
Integer
IP Address
JSON
Max
MIME Types (File)
Min
Not In
Numeric
Regular Expression
Required
Required If
Required Unless
Required With
Required With All
Required Without
Required Without All
Same
Size
String
Timezone
Unique (Database)
URL

Mostrar los errores

Se genera **$errors** como un elemento iterable con todos los errores que arroja Validator

```
{{ $errors->all() }}
```

Personalizar mensajes de errores

Traducción de mensajes de errores

Form request

1. Creamos el request
   `php artisan make:request Nombre`
2. Agregar validación en el método **rules** del archivo app/Http/Requests

```
return [
        'title' => 'required|unique:posts|max:255',
        'body' => 'required',
    ];
```

Personalizar mensajes

En tu request file cambias los mensajes con el método messages

```
public function messages()
{
    return [
        'title.required' => 'A title is required',
        'body.required'  => 'A message is required',
    ];
}
```

Sesión 8: Factory, Seeders y Autorización

1. Relaciones N:M
2. Factory
3. Faker
4. Seeders
5. Autorización

Actividad:

- Un ToDo tiene muchos Proyectos y viceversa
- Seeders para catálogos
- Limitar acciones por usuarios

Material

- Github
- Faker

Reglas de relaciones N:M

1. El nombre de la tabla intermedia es en el formato de **tabla1 - guion bajo - tabla2** en orden alfabético en singular
2. Los ids se nombran tal como ya se manejan **tabla_id**

Relaciones N:M

1. Crear tabla intermedia con las llaves foráneas
```
php artisan make:migration --create='tabla1_tabla2'
```

2. Si queremos integridad referencial poner los constraints
```
$table->foreign('todo_id')->references('id')->on('todos')->onDelete('cascade');
```

3. Corremos el migrate
```
php artisan migrate
```

Definimos la relación

Creamos el método de la relación

```
public function todos(){
    return $this->belongsToMany('Modelo');
}
```

Pero si queremos cambiarlo

```
belongsToMany('Modelo', 'tabla', 'llave_aqui',
'llave_alla');
```

Obtener los datos de la intermedia

- pivot es un atributo de mi resultado de la relación

```
$todo->projects()->first()->pivot->atributo
```

- Entonces tengo que poder definirlo en la relación, de otro modo, no puedo acceder a ellos

```
return $this->belongsToMany('Modelo')->withPivot('extra1', 'extra2');
```

Guardar los datos de la intermedia

- Attach agrega la nueva relación (registro)

```
$todo->projects()->first()->attach($id, ['tag'=>'asdf'])
```

- O agregar varios

```
$todo->projects()->first()->attach([id, id, id])
```

- También existe `saveMany($objeto)` pero no me permite enviar datos extras para la tabla intermedia

Eliminar datos de la relación

- Eliminar nueva relación (registro)

```
$todo->projects()->first()->detach($id)
```

Sincronizar relaciones

Existe la manera de agregar nuevos registros y quitar todos los demás.

```
$todo->projects()->first()->sync([id, id, id])
```

Solo quedarán en la tabla intermedia los ids indicados en el arreglo y todos los demás que hayan existido se eliminarán.

Factory

Factory es un patrón de diseño que está dedicado dedicado a la construcción de objetos de un subtipo de un tipo determinado

Creando factories

En database/factories/ModelFactory.php vemos un esquema de como usar un factory para usuarios.

Podemos definir los factories que queramos y nos sirvan para pruebas.

Definición de factories

```
$factory->define(Modelo::class, function () {
    return []
});
```

- Definimos modelos
- Podemos inyectar las dependencias que necesitamos
- Siempre regresar un arreglo asociativo

Faker

Clase externa que permite generar cadenas aleatorias de distintos tipos.

https://github.com/fzaninotto/Faker

Crear objetos con factory

Un solo objeto

```
$uno = factory(Modelo::class)->make();
```

Múltiples objetos

```
$muchos = factory(Modelo::class, cuantos)->make();
```

Creandolos y guardandolos

```
->create();
```

Creando un seeder

```
php artisan make:seeder Nombre
```

En el método run metemos todo lo que deseamos, ejemplo, un insert.

```
DB::table('users')->insert([
            'name' => str_random(10),
            'email' => str_random(10).'@gmail.com',
            'password' => bcrypt('secret'),
        ]);
```

Seeders con Factories

En nuestro seeder solo tenemos que mandar a llamar nuestro factory.

```
factory(App\User::class, 50)->create()
```

Ejecutar seeders

Todos los seeders

```
php artisan db:seed
```

Solo el de alguna clase

```
php artisan db:seed --class=UserTableSeeder
```

Autorización

Definir reglas para poder o no hacer cosas en nuestro sistema.

Se definen en el archivo app/Providers/AuthServiceProvider.php

Definir reglas

```
$gate->define('delete-task', function ($user, $task) {
            return ($user->id === $task->user_id &&
$task->status === true);
        });
```

Usando la regla

En el controlador valido con gate denies o allows

```
if (Gate::denies('accion-nombre', $param)) {
    abort(403);
}
```

O a través del usuario

```
if ($request->user()->cannot('accion-nombre', $param)) {
    abort(403);
}
```

Validando en las vistas

```
@can('update-post', $post)
    Bla bla
@else
    Algo mas
@endcan
```

Sesión 9: Scope, accesors y mutators

1. Scopes
2. Accessors
3. Mutators
4. Casts

Actividad:

- Aplicar filtro para ToDos

Material

- Github
-

Scopes

Los query scopes son la definición de un query que se usan mucho dentro de una aplicación.

Se define en el modelo

Su nombre solo antece el nombre Scope

Creando un scope

```
public function scopeActive($query, $arg)
    {
        return $query->where('active', $arg);
    }
```

Llamando a un scope

```
$users = App\User::active($arg)->orderBy('created_at')->get();
```

Accessors

La manera más fácil de entenderlos es pensar en los getters.

Pero lo primero es lo primero, ¿cuándo se deben usar los getters?

Para acceder a un atributo con un proceso de por medio

Reglas de los accessors

1. El método se declara en el modelo
2. El método comienza con get
3. El nombre del atributo va en camel case
4. El método termina con Attribute
5. El método recibe como argumento el atributo
6. Debe retornar el valor

Creando un accessor

```
public function getDuedateAttribute($value)
    {
        return ucfirst($value);
    }
```

Accediendo al elemento

```
$firstName = $user->first_name;
```

Mutators

La manera más fácil de entenderlos es pensar en los setters.

Pero lo primero es lo primero, ¿cuándo se deben usar los setters?

Para guardar un atributo que requiere de un proceso previo antes de guardarlo

Reglas de los accessors

1. El método se declara en el modelo
2. El método comienza con set
3. El nombre del atributo va en camel case
4. El método termina con Attribute
5. El método recibe el valor a guardar como argumento
6. Debe guardar el valor usando $this

Creando un mutator

```
public function setFirstNameAttribute($value)
    {
        $this->attributes['first_name'] = strtolower($value);
    }
```

Guardando un elemento

```
$user->first_name = 'Sally';
```

Casteo de atributos

Convertimos atributos del modelo a otro tipo de dato

```
protected $casts = [
        'is_admin' => 'boolean',
    ];
```

integer, real, float, double, string, boolean, object, array, collection, date y datetime

Clonar proyectos de Laravel

1. Crear archivo .env con la configuración deseada
2. Correr
   `composer install`
3. Crear la base de datos y configuraciones necesarias del proyecto
4. Correr los migrates
   `php artisan migrate`
5. Generar una key
   `php artisan key:generate`
6. Ejecutar los seeders
   `php artisan db:seed`

Sesión 10: Collections

1. Collections
2. Paginado
3. Dudas y Repaso

Actividad:

- Un

Material

- Github
- Collections

Paginado

Se usa el paginate(numero_de_elementos) en lugar del get o el all

No se puede paginar con relaciones directo en el query pero si fuera

En la vista se usa objeto->render()

Recomendaciones